(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) **World Intellectual Property Organization**
International Bureau

(43) **International Publication Date**
24 February 2011 (24.02.2011)

PCT

(10) **International Publication Number**
WO 2011/022555 A2

(51) **International Patent Classification**:
H03M 13/00 (2006.01) ***G06F 11/10*** (2006.01)
H03M 13/05 (2006.01)

(21) **International Application Number**:
PCT/US2010/046027

(22) **International Filing Date**:
19 August 2010 (19.08.2010)

(25) **Filing Language**: English

(26) **Publication Language**: English

(30) **Priority Data**:

61/235,285	19 August 2009 (19.08.2009)	US
12/604,773	23 October 2009 (23.10.2009)	US
61/257,146	2 November 2009 (02.11.2009)	US
61/353,910	11 June 2010 (11.06.2010)	US
12/859,161	18 August 2010 (18.08.2010)	US

(71) **Applicant** *(for all designated States except US)*: **QUALCOMM Incorporated** [US/US]; ATTN: International IP Administration, 5775 Morehouse Drive, San Diego, California 92121-1714 (US).

(72) **Inventors; and**
(75) **Inventors/Applicants** *(for US only)*: **LUBY, Michael, G.** [US/US]; 5775 Morehouse Drive, San Diego, California 92121-1714 (US). **SHOKROLLAHI, Mohammad, Amin** [US/CH]; 5775 Morehouse Drive, San Diego, California 92121-1714 (US). **MINDER, Lorenz** [CH/US]; 5775 Morehouse Drive, San Diego, California 92121-1714 (US).

(74) **Agent**: **JACOBS, Jeffrey, D.**; ATTN: International IP Administration, 5775 Morehouse Drive, San Diego, California 92121-1714 (US).

(81) **Designated States** *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PE, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) **Designated States** *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK,

[Continued on next page]

(54) **Title**: METHODS AND APPARATUS EMPLOYING FEC CODES WITH PERMANENT INACTIVATION OF SYMBOLS FOR ENCODING AND DECODING PROCESSES

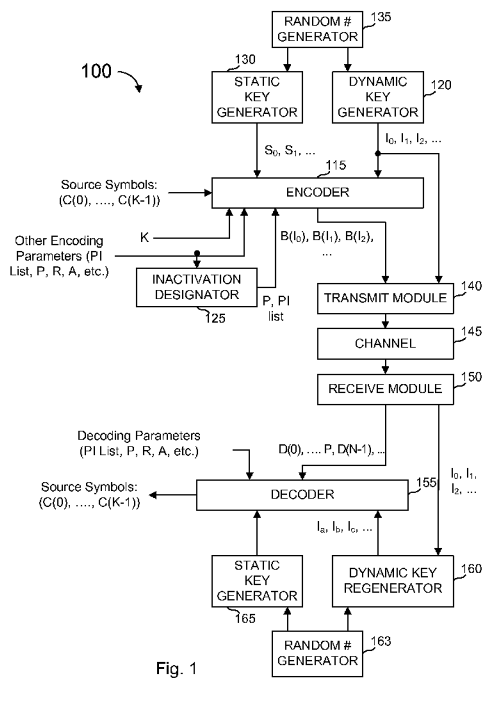


Fig. 1

(57) **Abstract**: Encoding of a plurality of encoded symbols is provided wherein an encoded symbol is generated from a combination of a first symbol generated from a first set of intermediate symbols and a second symbol generated from a second set of intermediate symbols, each set having at least one different coding parameter, wherein the intermediate symbols are generated based on the set of source symbols. A method of decoding data is also provided, wherein a set of intermediate symbols is decoded from a set of received encoded symbols, the intermediate symbols organized into a first and second sets of symbols for decoding, wherein intermediate symbols in the second set are permanently inactivated for the purpose of scheduling the decoding process to recover the intermediate symbols from the encoded symbols, wherein at least some of the source symbols are recovered from the decoded set of intermediate symbols.

METHODS AND APPARATUS EMPLOYING FEC CODES WITH PERMANENT INACTIVATION OF SYMBOLS FOR ENCODING AND DECODING PROCESSES

CROSS REFERENCES

[0001] This application is a continuation-in-part of U.S. Patent Application No. 12/604,773, filed October 23, 2009, naming M. Amin Shokrollahi, et al. and entitled "Method and Apparatus Employing FEC Codes with Permanent Inactivation of Symbols for Encoding and Decoding Processes" and further claims priority from the following provisional applications, each naming M. Amin Shokrollahi, et al. and each entitled "Method and Apparatus Employing FEC Codes with Permanent Inactivation of Symbols for Encoding and Decoding Processes": U.S. Provisional Patent Application No. 61/353,910, filed June 11, 2010, U.S. Provisional Patent Application No. 61/257,146, filed November 2, 2009, and U.S. Provisional Patent Application No. 61/235,285, filed August 19, 2009. Each provisional and nonprovisional application cited above is hereby incorporated by reference for all purposes.

[0002] The following references are herein incorporated by reference in their entirety for all purposes:

[0003] 1) U.S. Patent No. 6,307,487 issued to Michael G. Luby entitled "Information Additive Code Generator and Decoder for Communication Systems" (hereinafter "Luby I");

[0004] 2) U.S. Patent No. 6,320,520 issued to Michael G. Luby entitled "Information Additive Group Code Generator and Decoder for Communication Systems" (hereinafter "Luby II");

[0005] 3) U.S. Patent No. 7,068,729 issued to M. Amin Shokrollahi entitled "Multi-Stage Code Generator and Decoder for Communication Systems" (hereinafter "Shokrollahi I");

[0006] 4) U.S. Patent No. 6,856,263 issued to M. Amin Shokrollahi entitled "Systems and Processes for Decoding a Chain Reaction Code Through Inactivation" (hereinafter "Shokrollahi II");

[0007] 5) U.S. Patent No. 6,909,383, issued to M. Amin Shokrollahi entitled "Systematic Encoding and Decoding of Chain Reaction Codes" (hereafter "Shokrollahi III");

[0008] 6) U. S. Patent Publication No. 2006/0280254 naming Michael G. Luby and M. Amin Shokrollahi and entitled “In-Place Transformations with Applications to Encoding and Decoding Various Classes of Codes” (hereafter “Luby III”);

[0009] 7) U.S. Patent Publication No. 2007/0195894 naming M. Amin Shokrollahi and entitled “Multiple-Field Based Code Generator and Decoder for Communications Systems” (hereafter “Shokrollahi IV”).

FIELD OF THE INVENTION

[0010] The present invention relates to encoding and decoding data in communications systems and more specifically to communication systems that encode and decode data to account for errors and gaps in communicated data in an efficient manner.

BACKGROUND OF THE INVENTION

[0011] Techniques for transmission of files between a sender and a recipient over a communications channel are the subject of much literature. Preferably, a recipient desires to receive an exact copy of data transmitted over a channel by a sender with some level of certainty. Where the channel does not have perfect fidelity (which covers most all physically realizable systems), one concern is how to deal with data lost or garbled in transmission. Lost data (erasures) are often easier to deal with than corrupted data (errors) because the recipient cannot always tell when corrupted data is data received in error. Many error-correcting codes have been developed to correct for erasures and/or for errors. Typically, the particular code used is chosen based on some information about the infidelities of the channel through which the data is being transmitted and the nature of the data being transmitted. For example, where the channel is known to have long periods of infidelity, a burst error code might be best suited for that application. Where only short, infrequent errors are expected a simple parity code might be best.

[0012] As used herein, “source data” refers to data that is available at one or more senders and that a receiver is used to obtain, by recovery from a transmitted sequence with or without errors and/or erasures, etc. As used herein, “encoded data” refers to data that is conveyed and can be used to recover or obtain the source data. In a simple case, the encoded data is a copy of the source data, but if the received encoded data differs (due to errors and/or erasures) from the transmitted encoded data, in this simple case the source data might not be entirely recoverable absent additional data about the source data. Transmission can be through space or time. In a more complex case, the encoded data is generated based on source data in a transformation and is transmitted from one or more senders to receivers. The encoding is said to be "systematic" if the source data is found to be part of the encoded data. In a simple example of systematic encoding, redundant information about the source data is appended to the end of the source data to form the encoded data.

[0013] Also as used herein, "input data" refers to data that is present at an input of an FEC (forward-error correcting) encoder apparatus or an FEC encoder module, component, step, etc., ("FEC encoder") and "output data" refers to data that is present at an output of an FEC encoder. Correspondingly, output data would be expected to be present at an input of an FEC decoder and the FEC decoder would be expected to output the input data, or a correspondence thereof, based on the output data it processed. In some cases, the input data is, or includes, the source data, and in some cases, the output data is, or includes, the encoded data. In other cases, a sender device or sender program code may comprise more than one FEC encoder, i.e., source data is transformed into encoded data in a series of a plurality of FEC encoders. Similarly at the receiver, there may be more than one FEC decoder applied to generate source data from received encoded data.

[0014] Data can be thought of as partitioned into symbols. An encoder is a computer system, device, electronic circuit, or the like, that generates encoded symbols or output symbols from a sequence of source symbols or input symbols and a decoder is the counterpart that recovers a sequence of source symbols or input symbols from received or recovered encoded symbols or output symbols. The encoder and decoder are separated in time and/or space by the channel and any received encoded symbols might not be exactly the same as corresponding transmitted encoded symbols and they might not be received in exactly the same sequence as they were transmitted. The "size" of a symbol can be measured in bits, whether or not the symbol is actually broken into a bit stream, where a symbol has a size of M bits when the symbol is selected from an alphabet of 2^M symbols. In many of the examples herein, symbols are measured in bytes and codes might be over a field of 256 possibilities (there are 256 possible 8-bit patterns), but it should be understood that different units of data measurement can be used and it is well-known to measure data in various ways.

[0015] Luby I describes the use of codes, such as chain reaction codes, to address error correction in a compute-efficient, memory-efficient and bandwidth-efficient manner. One property of the encoded symbols produced by a chain reaction encoder is that a receiver is able to recover the original file as soon as enough encoded symbols have been received.

Specifically, to recover the original K source symbols with a high probability, the receiver needs approximately K+A encoded symbols.

[0016] The "absolute reception overhead" for a given situation is represented by the value A, while a "relative reception overhead" can be calculated as the ratio A/K. The absolute reception overhead is a measure of how much extra data needs to be received beyond the information theoretic minimal amount of data, and it may depend on the reliability of the decoder and may vary as a function of the number, K, of source symbols. Similarly, the relative reception overhead, A/K, is a measure of how much extra data needs to be received beyond the information theoretic minimal amount of data relative to the size of the source data being recovered, and also may depend on the reliability of the decoder and may vary as a function of the number K of source symbols.

[0017] Chain reaction codes are extremely useful for communication over a packet based network. However, they can be fairly computationally intensive at times. A decoder might be able to decode more often, or more easily, if the source symbols are encoded using a static encoder prior to a dynamic encoder that encodes using a chain reaction or another rateless code. Such decoders are shown in Shokrollahi I, for example. In examples shown there, source symbols are input symbols to a static encoder that produces output symbols that are input symbols to a dynamic encoder that produces output symbols that are the encoded symbols, wherein the dynamic encoder is a rateless encoder that that can generate a number of output symbols in a quantity that is not a fixed rate relative to the number of input symbols. The static encoder might include more than one fixed rate encoder. For example a static encoder might include a Hamming encoder, a low-density parity-check ("LDPC") encoder, a high-density parity-check ("HDPC") encoder, and/or the like.

[0018] Chain reaction codes have a property that as some symbols are recovered at the decoder from the received symbols, those symbols might be able to be used to recover additional symbols, which in turn might be used to recover yet more symbols. Preferably, the chain reaction of symbol solving at the decoder can continue such that all of the desired symbols are recovered before the pool of received symbols is used up. Preferably, the computational complexity of performing chain reaction encoding and decoding processes is low.

[0019] A recovery process at the decoder might involve determining which symbols were received, creating a matrix that would map the original input symbols to those encoded symbols that were received, then inverting the matrix and performing a matrix multiplication of the inverted matrix and a vector of the received encoded symbols. In a typical system, a brute force implementation of this can consume excessive computing effort and memory requirements. Of course, for a particular set of received encoded symbols, it might be impossible to recover all of the original input symbols, but even where it is possible, it might be very computationally expensive to compute the result.

[0020] Shokrollahi II describes an approach called “inactivation”, wherein decoding occurs in two steps. In the first step, the decoder takes stock of what received encoded symbols it has available, what the matrix might look like and determines, at least approximately, a sequence of decoding steps that would allow for the chain reaction process to complete given the received encoded symbols. In the second step, the decoder runs the chain reaction decoding according to the determined sequence of decoding steps. This can be done in a memory-efficient manner (i.e., a manner that requires less memory storage for the operation than a more memory-inefficient process).

[0021] In an inactivation approach, the first decoding step involves manipulating the matrix, or its equivalent, to determine some number of input symbols that can be solved for and when the determination stalls, designating one of the input symbols as an “inactivated symbol” and continue the determination process assuming that the inactivated symbol is indeed solved, then at the end, solving for the inactivated symbols using Gaussian elimination or some other method to invert a matrix that is much smaller than the original decoding matrix. Using that determination, the chain reaction sequence can be performed on the received encoded symbols to arrive at the recovered input symbols, which can either be all of the original input symbols or a suitable set of the original input symbols.

[0022] For some applications that impose tight constraints on the decoder, such as where the decoder is in a low-power device with limited memory and computing power, or such as when there are tight constraints on the allowable absolute or relative reception overhead, improved methods might be indicated relative to the inactivation approach described above.

[0023] Also, methods for partitioning a file or large block of data into as few source blocks as possible subject to a constraint on the smallest sub-symbol size, and then subject to this split into as few sub-blocks as possible subject to a constraint on the maximum sub-block size, might be useful.

BRIEF SUMMARY OF THE INVENTION

[0024] According to one embodiment of an encoder according to aspects of the present invention, an encoder, at, in or for a sender that transmits an ordered set of source symbols from one or more senders to one or more receivers over a communications channel, wherein the encoder generates data to be sent that includes a plurality of encoded symbols generated from the source symbols. In a first step, intermediate symbols are generated from the source symbols using a method that is invertible, i.e., there is also an inverse method for generating the source symbols from the intermediate symbols. In another step, the intermediate symbols are partitioned into a first set of intermediate symbols and a second set of intermediate symbols, wherein there is at least one intermediate symbol in the first set of intermediate symbols and there is at least one intermediate symbol in the second set of intermediate symbols and at least one encoded symbol is generated from at least one intermediate symbol from each of the two sets. In some variations, there are more than two sets.

[0025] In some embodiments, values for a first set and a second set of temporary symbols are generated, wherein the values of the first set of temporary symbols depend on the values of the first set of intermediate symbols and the values for the second set of temporary symbols depend on the values of the second set of intermediate symbols. The values for encoded symbols are generated from the first set and the second set of temporary symbols.

[0026] In some variations, the number of encoded symbols that can be generated is independent of the number of source symbols.

[0027] Decoder embodiments are also provided. According to one embodiment of a decoder according to aspects of the present invention, a decoder, at, in or for a receiver, receives encoded symbols generated from intermediate symbols, wherein the intermediate symbols are generated from source symbols using a method that is invertible, i.e., there is also an inverse method for generating the source symbols from the intermediate symbols, and wherein at least one of the intermediate symbols is designated a permanently inactivated symbol and wherein there is at least another one of the intermediate symbols that is not among the permanently inactivated symbols. The decoder decodes, from the received encoded symbols, a set of intermediate symbols and the decoder takes into account at least one permanently inactivated symbol, and generates source symbols from the decoded set of intermediate symbols using the inverse method.

[0028] In decoding, decoding steps are scheduled, setting aside the scheduling of permanently inactivated symbols. The permanently inactivated symbols can be solved using novel or conventional methods and then used in solving for the other intermediate symbols. One approach to solving for the permanent inactivated symbols (and other on-the-fly inactivations, if used) might be by applying Gaussian elimination to solve for the inactivated symbols. Some of the remaining intermediate symbols are recovered based on the values of the recovered permanently inactivated symbols and received encoded symbols.

[0029] In some variations of the decoding method, the permanently inactivated symbols comprise the second set of intermediate symbols from the encoding embodiments. In some variations of the decoding method, the permanently inactivated symbols comprise a subset of the intermediate symbols wherein the corresponding encoding method is not a multi-stage chain reaction code. Such encoding methods might include one or more of a Tornado code, a Reed-Solomon code, a chain reaction code (examples described in Luby I), or the like for the subset of the intermediate symbols.

[0030] Intermediate symbols are used for encoding and decoding, wherein the method for generating intermediate symbols from source symbols and the corresponding inverse method, are indicated for a desired set of performance characteristics, such as decodability. In some embodiments, the intermediate symbols comprise the source symbols. In some embodiments, the intermediate symbols comprise the source symbols, along with redundant symbols that are generated from the source symbols, where the redundant symbols might be chain reaction symbols, LDPC symbols, HDPC symbols or other types of redundant symbols. Alternatively, intermediate symbols could be based on prescribed relationships between symbols, for example relationships between the intermediate symbols and the source symbols, and additional LDPC and HDPC relationships among the intermediate symbols, wherein a decoding method is used to generate the intermediate symbols from the source symbols based on the prescribed relationships.

[0031] The methods and systems can be implemented by electronic circuits or by a processing device that executes programming instructions and has the appropriate instruction program code to implement encoding and/or decoding.

[0032] Numerous benefits are achieved by way of the present invention. For example, in a specific embodiment, the computational expense of encoding data for transmission over a channel is reduced. In another specific embodiment, the computational expense of decoding such data is reduced. In another specific embodiment, the absolute and relative reception overhead is substantially reduced. Depending upon the embodiment, one or more of these benefits may be achieved. These and other benefits are provided in more detail throughout the present specification and more particularly below.

[0033] A further understanding of the nature and the advantages of the inventions disclosed herein may be realized by reference to the remaining portions of the specification and the attached drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

[0034] Fig. 1 is a block diagram of a communications system that uses multi-stage coding that includes permanent inactivation, along with other features and elements.

[0035] Fig. 2 is a table of variables, arrays and the like, that are used in various other figures herein.

[0036] Fig. 3 is a block diagram of a specific embodiment of the encoder shown in Fig. 1.

[0037] Fig. 4 is a block diagram showing the dynamic encoder of Fig. 3 in greater detail.

[0038] Fig. 5 is a flowchart illustrating a permanent inactivation (PI) encoding process.

[0039] Fig. 6 is a flowchart illustrating a dynamic encoding process.

[0040] Fig. 7 is a flowchart of an operation of calculating a weight for a symbol calculation.

[0041] Fig. 8 illustrates a table that might be stored in memory, usable to determine a degree of a symbol based on a lookup value.

[0042] Fig. 9 shows a matrix used in an encoding or decoding process.

[0043] Fig. 10 shows an equation representing parts of the matrix shown in Fig. 9, for a specific minimal polynomial.

[0044] Fig. 11 is a flowchart illustrating a process for setting up an array for use in encoding or decoding.

[0045] Fig. 12 illustrates a matrix representation of a set of equations to be solved by a decoder to recover an array, C(), representing recovered source symbols from an array, D(), representing received encoded symbols, using a submatrix SE representing R static symbols or equations known by the decoder.

[0046] Fig. 13 illustrates a matrix resulting from row/column permutations of the matrix of Fig. 12, using OTF inactivation.

[0047] Fig. 14 is a block diagram describing a process for generating the matrix in Fig. 12.

[0048] Fig. 15 illustrates a matrix representation of a set of equations to be solved by a decoder to recover an array, C(), representing recovered source symbols from an array, D(), representing received encoded symbols, using a submatrix SE and a submatrix corresponding to permanently inactivated symbols.

[0049] Fig. 16 is a flowchart illustrating a process for generating an LT submatrix as might be used in the matrix of Fig. 12 or the matrix of Fig. 15.

[0050] Fig. 17 is a flowchart illustrating a process for generating a PI submatrix as might be used in the matrix of Fig. 15.

[0051] Fig. 18 is a block diagram of a matrix generator.

[0052] Fig. 19 is a flowchart illustrating a process for generating an SE submatrix.

[0053] Fig. 20 is a flowchart illustrating a process for generating a PI submatrix.

[0054] Fig. 21 is a flowchart illustrating a process for solving for recovered symbols in a decoder.

[0055] Fig. 22 illustrates a matrix representation of a set of equations to be solved by a decoder to recover an array, C(), representing recovered source symbols from an array, D(), representing received encoded symbols, after permutations.

[0056] Fig. 23 illustrates a matrix representation of a set of equations to be solved by a decoder and corresponding to the matrix shown in Fig. 26.

[0057] Fig. 24 illustrates a matrix representation usable as part of a decoding process.

[0058] Fig. 25 illustrates a matrix representation usable as another part of a decoding process.

[0059] Fig. 26 illustrates a matrix representation of a set of equations to be solved by a decoder after partial solution.

[0060] Fig. 27 is a flowchart illustrating another process for solving for recovered symbols in a decoder.

[0061] Fig. 28 illustrates a matrix representation of a set of equations to be solved by a decoder.

[0062] Fig. 29 illustrates a matrix representation of a set of equations to be solved by a decoder.

[0063] Fig. 30 illustrates an example encoding system that might be implemented as hardware modules, software modules, or portions of program code stored in a program store and executed by a processor, possibly as a collective unit of code not separated as shown in the figure.

[0064] Fig. 31 illustrates an example decoding system that might be implemented as hardware modules, software modules, or portions of program code stored in a program store and executed by a processor, possibly as a collective unit of code not separated as shown in the figure.

[0065] Attached as Appendix A is a code specification for a specific embodiment of an encoder/decoder system, an error correction scheme, and applications to reliable delivery of data objects, sometimes with details of the present invention used, which also includes a specification of how a systematic encoder/decoder might be used in object delivery transport. It should be understood that the specific embodiments described in Appendix A are not limiting examples of the invention and that some aspects of the invention might use the teachings of Appendix A while others might not. It should also be understood that limiting statements in Appendix A may be limiting as to requirements of specific embodiments and such limiting statements might or might not pertain the claimed inventions and, therefore, the claim language need not be limited by such limiting statements.

DETAILED DESCRIPTION OF SPECIFIC EMBODIMENTS

[0066] Details for implementing portions of encoders and decoders that are referenced herein are provided by Luby I, Luby II, Shokrollahi I, Shokrollahi II, Shokrollahi III, Luby III, and Shokrollahi IV and are not entirely repeated here for the sake of brevity. The entire disclosures of those are herein incorporated by reference for all purposes and it is to be understood that the implementations therein are not required of the present invention, and many other variations, modifications, or alternatives can also be used, unless otherwise indicated.

[0067] Multi-stage encoding, as described herein, encodes the source data in a plurality of stages. Typically, but not always, a first stage adds a predetermined amount of redundancy to the source data. A second stage then uses a chain reaction code, or the like, to produce encoded symbols from the original source data and the redundant symbols computed by the first stage of the encoding. In one specific embodiment, the received data is first decoded using a chain reaction decoding process. If that process is not successful in recovering the original data completely, a second decoding step can be applied.

[0068] Some of the embodiments taught herein can be applied to many other types of codes, for example to the codes as described in the Internet Engineering Task Force (IETF) Request for Comments (RFC) 5170 (hereinafter "IETF LDPC codes"), and to the codes described in U.S. Patent Nos. 6,073,250, 6,081,909 and 6,163,870 (hereinafter "Tornado codes"), resulting in improvements in reliability and/or CPU and/or memory performance for those types of codes.

[0069] One advantage of some embodiments taught herein, is that fewer arithmetic operations are necessary to produce encoded symbols, as compared to chain reaction coding alone. Another advantage of some specific embodiments that include a first stage of encoding and a second stage of encoding is that the first stage of encoding and the second stage of encoding can be done at separate times and/or by separate devices, thus partitioning the computational load and minimizing the overall computational load and also the memory size and access pattern requirements. In embodiments of multi-stage encoding, redundant symbols are generated from the input file during the first stage of encoding. In these embodiments, in the second stage of encoding, encoded symbols are generated from the combination of the input file and the redundant symbols. In some of these embodiments, the encoded symbols can be generated as needed. In embodiments in which the second stage comprises chain reaction encoding, each encoded symbol can be generated without regard to how other encoded symbols are generated. Once generated, these encoded symbols can then be placed into packets and transmitted to their destination, with each packet containing one or more encoded symbols. Non-packetized transmission techniques can be used instead or as well.

[0070] As used herein, the term "file" refers to any data that is stored at one or more sources and is to be delivered as a unit to one or more destinations. Thus, a document, an image, and a file from a file server or computer storage device, are all examples of "files" that can be delivered. Files can be of known size (such as a one megabyte image stored on a hard disk) or can be of unknown size (such as a file taken from the output of a streaming source). Either way, the file is a sequence of source symbols, where each source symbol has a position in the file and a value. A "file" may also be used to refer to a short portion of a streaming source, i.e., the stream of data may be partitioned into one second intervals, and the block of source data within each such one second interval may be considered to be a "file". As another example, the blocks of data from a video streaming source may be further partitioned into multiple parts based on priorities of that data defined for example by a video system that can playout the video stream, and each part of each block may be considered to be a "file". Thus, the term "file" is used generally and is not intended to be extensively limiting.

[0071] As used herein, source symbols represent the data that is to be transmitted or conveyed, and encoded symbols represent the data generated based on source symbols that is conveyed over a communications network, or stored, to enable the reliable reception and/or regeneration of the source symbols. Intermediate symbols represent symbols that are used or generated during an intermediate step of the encoding or decoding processes, wherein typically there is a method for generating intermediate symbols from source symbols and a corresponding inverse method for generating the source symbols from the intermediate symbols. Input symbols represent data that is input to one or more steps during the process of encoding or decoding, and output symbols represent data that is output from one or more steps during the process of encoding or decoding.

[0072] In many embodiments, these different types or labels of symbols can be the same or comprised at least partially of other types of symbols, and in some examples the terms are used interchangeably. In an example, suppose that a file to be transmitted is a text file of 1,000 characters, each of which is deemed a source symbol. If those 1,000 source symbols are provided as is to an encoder, that in turn outputs encoded symbols that are transmitted, the source symbols are also input symbols. However, in embodiments where the 1,000 source symbols are in a first step converted to 1,000 (or more or fewer) intermediate symbols and the intermediate symbols are provided to the encoder to generate encoded symbols in a second step, the source symbols are the input symbols and the intermediate symbols are the output symbols in the first step, and the intermediate symbols are the input symbols and the encoded symbols are the output symbols in the second step, whereas the source symbols are the overall input symbols to this two-step encoder and the encoded symbols are the overall output symbols of this two-step encoder. If, in this example, the encoder is a systematic encoder, then the encoded symbols may comprise the source symbols together with repair symbols generated from the intermediate symbols, whereas the intermediate symbols are distinct from both the source symbols and the encoded symbols. If instead, in this example, the encoder is a non-systematic encoder, then the intermediate symbols may comprise the source symbols together with redundant symbols generated from the source symbols, using for example an LDPC and/or HDPC encoder in the first step, whereas the encoded symbols are distinct from both the source symbols and the intermediate symbols.

[0073] In other examples, there are more symbols and each symbol represents more than one character. In either case, where there is a source-to-intermediate symbol conversion in a transmitter, the receiver might have a corresponding intermediate-to-source symbol conversion as the inverse.

[0074] Transmission is the process of transmitting data from one or more senders to one or more recipients through a channel in order to deliver a file. A sender is also sometimes referred to as the encoder. If one sender is connected to any number of recipients by a perfect channel, the received data can be an exact copy of the source file, as all the data will be received correctly. Here, we assume that the channel is not perfect, which is the case for most real-world channels. Of the many channel imperfections, two imperfections of interest are data erasure and data incompleteness (which can be treated as a special case of data erasure). Data erasure occurs when the channel loses or drops data. Data incompleteness occurs when a recipient does not start receiving data until some of the data has already passed it by, the recipient stops receiving data before transmission ends, the recipient chooses to only receive a portion of the transmitted data, and/or the recipient intermittently stops and starts again receiving data. As an example of data incompleteness, a moving satellite sender might be transmitting data representing a source file and start the transmission before a recipient is in range. Once the recipient is in range, data can be received until the satellite moves out of range, at which point the recipient can redirect its satellite dish (during which time it is not receiving data) to start receiving the data about the same input file being transmitted by another satellite that has moved into range. As should be apparent from reading this description, data incompleteness is a special case of data erasure, since the recipient can treat the data incompleteness (and the recipient has the same problems) as if the recipient was in range the entire time, but the channel lost all the data up to the point where the recipient started receiving data. Also, as is well known in communication systems design, detectable errors can be considered equivalent to erasures by simply dropping all data blocks or symbols that have detectable errors.

[0075] In some communication systems, a recipient receives data generated by multiple senders, or by one sender using multiple connections. For example, to speed up a download, a recipient might simultaneously connect to more than one sender to transmit data concerning the same file. As another example, in a multicast transmission, multiple multicast data streams might be transmitted to allow recipients to connect to one or more of these streams to match the aggregate transmission rate with the bandwidth of the channel connecting them to the sender. In all such cases, a concern is to ensure that all transmitted data is of independent use to a recipient, i.e., that the multiple source data is not redundant among the streams, even when the transmission rates are vastly different for the different streams, and when there are arbitrary patterns of loss.

[0076] In general, a communication channel is that which connects the sender and the recipient for data transmission. The communication channel could be a real-time channel, where the channel moves data from the sender to the recipient as the channel gets the data, or the communication channel might be a storage channel that stores some or all of the data in its transit from the sender to the recipient. An example of the latter is disk storage or other storage device. In that example, a program or device that generates data can be thought of as the sender, transmitting the data to a storage device. The recipient is the program or device that reads the data from the storage device. The mechanisms that the sender uses to get the data onto the storage device, the storage device itself and the mechanisms that the recipient uses to get the data from the storage device collectively form the channel. If there is a chance that those mechanisms or the storage device can lose data, then that would be treated as data erasure in the communication channel.

[0077] When the sender and recipient are separated by a communication channel in which symbols can be erased, it is preferable not to transmit an exact copy of an input file, but instead to transmit data generated from the input file that assists with recovery of erasures. An encoder is a circuit, device, module or code segment that handles that task. One way of viewing the operation of the encoder is that the encoder generates encoded symbols from source symbols, where a sequence of source symbol values represent the input file. Each source symbol would thus have a position, in the input file, and a value. A decoder is a circuit, device, module or code segment that reconstructs the source symbols from the encoded symbols received by the recipient. In multi-stage coding, the encoder and the decoder are sometimes further divided into sub-modules each performing a different task.

[0078] In embodiments of multi-stage coding systems, the encoder and the decoder can be further divided into sub-modules, each performing a different task. For instance, in some embodiments, the encoder comprises what is referred to herein as a static encoder and a dynamic encoder. As used herein, a "static encoder" is an encoder that generates a number of redundant symbols from a set of source symbols, wherein the number of redundant symbols is determined prior to encoding. When static encoding is used in a multi-stage coding system, the combination of the source symbols and the redundant symbols generated from the source symbols using a static encoder are often referred to as the intermediate symbols. Examples of potential static encoding codes include Reed-Solomon codes, Tornado codes, Hamming codes, LDPC codes such as the IETF LDPC codes, etc. The term "static decoder" is used herein to refer to a decoder that can decode data that was encoded by a static encoder.

[0079] As used herein, a "dynamic encoder" is an encoder that generates encoded symbols from a set of input symbols, where the number of possible encoded symbols is independent of the number of input symbols, and where the number of encoded symbols to be generated need not be fixed. Often in a multi-stage code, the input symbols are the intermediate symbols generated using a static code and the encoded symbols are generated from intermediate symbols using a dynamic encoder. One example of a dynamic encoder is a chain reaction encoder, such as the encoders taught in Luby I and Luby II. The term "dynamic decoder" is used herein to refer to a decoder that can decode data that was encoded by a dynamic encoder.

[0080] In some embodiments, encoding that is multi-stage code and systematic uses a decoding process applied to the source symbols to obtain the intermediate symbol values based on the relationships defined by the static encoder among the intermediate symbols and defined by the dynamic encoder between the intermediate symbols and the source symbols, and then a dynamic encoder is used to generate additional encoded symbols, or repair symbols, from the intermediate symbols. Similarly, a corresponding decoder has a decoding process to receive encoded symbols and decode from them the intermediate symbol values based on the relations defined by the static encoder among the intermediate symbols and defined by the dynamic encoder between the intermediate symbols and the received encoded symbols, and then a dynamic encoder is used to generate any missing source symbols from the intermediate symbols.

[0081] Embodiments of multi-stage coding need not be limited to any particular type of symbol. Typically, the values for the symbols are selected from an alphabet of 2^M symbols for some positive integer M. In such cases, a source symbol can be represented by a sequence of M bits of data from the input file. The value of M is often determined based on, for example, the uses of the application, the communication channel, and/or the size of the encoded symbols. Additionally, the size of an encoded symbol is often determined based on the application, the channel, and/or the size of the source symbols. In some cases, the coding process might be simplified if the encoded symbol values and the source symbol values were the same size (i.e., representable by the same number of bits or selected from the same alphabet). If that is the case, then the source symbol value size is limited when the encoded symbol value size is limited. For example, it may be desired to put encoded symbols in packets of limited size. If some data about a key associated with the encoded symbols were to be transmitted in order to recover the key at the receiver, the encoded symbol would preferably be small enough to accommodate, in one packet, the encoded symbol value and the data about the key.

[0082] As an example, if an input file is a multiple megabyte file, the input file might be broken into thousands, tens of thousands, or hundreds of thousands of source symbols with each source symbol encoding thousands, hundreds, or only few bytes. As another example, for a packet-based Internet channel, a packet with a payload of size of 1024 bytes might be appropriate (a byte is 8 bits). In this example, assuming each packet contains one encoded symbol and 8 bytes of auxiliary information, an encoded symbol size of 8128 bits ((1024 - 8) * 8) would be appropriate. Thus, the source symbol size could be chosen as M = (1024 - 8) * 8, or 8128 bits. As another example, some satellite systems use the MPEG packet standard, where the payload of each packet comprises 188 bytes. In that example, assuming each packet contains one encoded symbol and 4 bytes of auxiliary information, an encoded symbol size of 1472 bits ((188 - 4) * 8), would be appropriate. Thus, the source symbol size could be chosen as M = (188 - 4) * 8, or 1472 bits. In a general-purpose communication system using multi-stage coding, the application-specific parameters, such as the source symbol size (i.e., M, the number of bits encoded by a source symbol), might be variables set by the application.

[0083] Each encoded symbol has a value. In one preferred embodiment, which we consider below, each encoded symbol also has associated therewith an identifier called its "key." Preferably, the key of each encoded symbol can be easily determined by the recipient to allow the recipient to distinguish one encoded symbol from other encoded symbols. Preferably, the key of an encoded symbol is distinct from the keys of all other encoded symbols. There are various forms of keying discussed in previous art. For example, Luby I describes various forms of keying that can be employed in embodiments of the present invention. In other preferred embodiments, such as the one described in Appendix A, the key for an encoded symbol is referred to as an "Encoded Symbol Identifier", or "Encoding Symbol Identifier", or more simply the "ESI".

[0084] Multi-stage coding is particularly useful where there is an expectation of data erasure or where the recipient does not begin and end reception exactly when a transmission begins and ends. The latter condition is referred to herein as "data incompleteness." Regarding erasure events, multi-stage coding shares many of the benefits of chain reaction coding taught in Luby I. In particular, multi-stage encoded symbols are information additive, so any suitable number of packets can be used to recover an input file to a desired degree of accuracy. These conditions do not adversely affect the communication process when multi-stage coding is used, because the encoded symbols generated with multi-stage coding are information additive. For example, if a hundred packets are lost due to a burst of noise causing data erasure, an extra hundred packets can be picked up after the burst to replace the loss of the erased packets. If thousands of packets are lost because a receiver did not tune into a transmitter when it began transmitting, the receiver could just pickup those thousands of packets from any other period of transmission, or even from another transmitter. With multi-stage coding, a receiver is not constrained to pickup any particular set of packets, so it can receive some packets from one transmitter, switch to another transmitter, lose some packets, miss the beginning or end of a given transmission and still recover an input file. The ability to join and leave a transmission without receiver-transmitter coordination helps to simplify the communication process.

[0085] In some embodiments, transmitting a file using multi-stage coding can include generating, forming or extracting source symbols from an input file, computing redundant symbols, encoding source and redundant symbols into one or more encoded symbols, where each encoded symbol is generated based on its key independently of all other encoded symbols, and transmitting the encoded symbols to one or more recipients over a channel.

Additionally, in some embodiments, receiving (and reconstructing) a copy of the input file using multi-stage coding can include receiving some set or subset of encoded symbols from one of more data streams, and decoding the source symbols from the values and keys of the received encoded symbols.

Systematic Codes and Nonsystematic Codes

[0086] A systematic code is a code where the source symbols are among the encoded symbols that can be transmitted. In this case, the encoded symbols are comprised of source symbols and redundant symbols, also called repair symbols, generated from the source symbols. A systematic code is preferable over a non-systematic code for many applications, for a variety of reasons. For example, in a file delivery application, it is useful to be able to start transmitting data in sequential order while the data is being used to generate repair data, where the process of generating repair data can take some amount of time. As another example, many applications prefer to send the original source data in sequential order in its unmodified form to one channel, and to send repair data to another channel. One typical reason for this is to support both legacy receivers that don't incorporate FEC decoding while at the same time providing a better experience to enhanced receivers that do incorporate FEC decoding, wherein legacy receivers join only the source data channel and enhanced receivers join both the source data channel and the repair data channel.

[0087] In these and related types of applications it can sometimes be the case that the loss patterns and the fraction of loss among received source symbols by a receiver is quite different than that experienced among received repair symbols. For example, when source symbols are sent prior repair symbols, due to the bursty loss conditions of the channel, the fraction and pattern of loss among source symbols can be quite different than the corresponding fraction and pattern of loss among repair symbols, and the pattern of loss among source symbols may be far from what might be typical than if the loss were uniformly random. As another example, when the source data is sent on one channel and the repair data on another channel, there might be quite different loss conditions on the two channels. Thus, it is desirable to have a systematic FEC code that works well under different types of loss conditions.

[0088] Although examples herein refer to systematic codes (where the output or encoded symbols include the source or input symbols) or nonsystematic codes, the teachings herein should be assumed to be applicable to both, unless otherwise indicated. Shokrollahi III teaches methods to convert a non-systematic chain reaction code to a systematic code in such a way that the robustness properties of the non-systematic code are maintained by the systematic code so constructed.

[0089] In particular, using the methods taught in Shokrollahi III, the constructed systematic code has the property that there is little differentiation in terms of recoverability by the decoder between lost source symbols and lost repair symbols, i.e., the decoding recovery probability is essentially the same for a given amount of total loss almost independent of the proportion of the loss among the source symbols compared to the proportion of the loss among the repair symbols. Furthermore, the pattern of loss among the encoded symbols does not significantly affect the decoding recovery probability. In comparison, for the constructions of other systematic codes, such as those described for Tornado codes or for IETF LDPC codes, there is in many cases a strong differentiation in terms of recoverability by the decoder between lost source symbols and lost repair symbols, i.e., the decoding recovery probability can vary widely for the same for a given amount of total loss depending on the proportion of the loss among the source symbols compared to the proportion of the loss among the repair symbols. Furthermore, the pattern of loss among the encoded symbols can have a strong effect on the decoding recovery probability. The Tornado codes and IETF LDPC codes have reasonably good recovery properties if the losses of encoded symbols are uniformly random among all of the encoded symbols, but the recovery properties deteriorate as the loss model deviates from uniform random loss. Thus, in this sense, the embodiments taught in Shokrollahi III have advantages over other constructions of systematic codes.

[0090] For an FEC code with the property that there is a strong effect in terms of recoverability by the decoder depending on the proportion of lost source symbols and lost repair symbols, and depending on loss patterns, one approach to overcome this property when it is applicable is to send the encoded symbols in a uniformly random order, i.e., the combination of source and repair symbols are sent in uniformly random order, and thus the source symbols are randomly interspersed among the repair symbols. Sending the encoded symbols in random order has an advantage that whatever the channel loss model, whether the losses are bursty or uniformly random or some other type of losses, the losses to the encoded symbols are still random. However, as noted above, this approach is not desirable for some applications, e.g., for applications where it is desirable to send the source symbols in sequence before the repair symbols, or where the source symbols are sent on a different channel than the repair symbols.

[0091] In such cases, constructions of systematic codes where the pattern of loss among the encoded symbols does not greatly affect the recovery properties of the decoder are desired and some examples are provided herein.

[0092] As used herein, "random" and "pseudorandom" are often equivalent and/or interchangeable and may depend on context. For example, random losses may refer to which symbols are lost by a channel, which may truly be a random event, whereas a random selection of symbol neighbors might actually be a repeatable pseudorandom selection according to a nonrandom process, but that has the same or similar properties or behaviors as would be the case with a truly random selection. Unless otherwise indicated explicitly or by context, characterizing something as random is not meant to exclude pseudorandomness.

[0093] In one approach to such a systematic FEC encoder, source symbols are obtained by an encoder that includes multiple encoder sub-blocks or subprocesses, one of which operates as a decoder to generate intermediate symbols that are input symbols for another sub-block or subprocess. The intermediate symbols are then applied to another sub-block or subprocess that encodes the intermediate symbols into the encoded symbols so that the encoded symbols include the source symbols (along with additional, redundant symbols) generated from one consistent process, thereby providing robustness benefits and other benefits over an encoder that is a systematic encoder that uses one process (e.g., copying) to get the source symbols for the encoded symbol set and another process to get the redundant symbols for the encoded symbol set.

[0094] The output encoding can be a chain reaction encoder, a static encoder or other variations. Appendix A describes a systematic code embodiment. After reading the present disclosure, one of ordinary skill in the art should be able to easily extend the teachings of Shokrollahi III to apply to systematic codes such as the Tornado codes and IETF LDPC codes, to yield new versions of these codes that are also systematic codes but have better recovery properties. In particular, the new versions of these codes, obtained by applying the general method described below, are enhanced to have the property that the proportion of loss among the source symbols compared to the proportion of loss among the repair symbols does not significantly affect the decoding recovery probability, and furthermore that the pattern of loss does not significantly affect the decoding recovery probability. Thus, these codes can be effectively used in the applications described above that require usage of systematic FEC codes with recovery properties that are not strongly affected by different fractional loss amounts among source and repair symbols or by different loss patterns.

[0095] The new encoding method can be applied generally to encoding for systematic FEC codes, non-systematic FEC codes, fixed rate FEC codes and chain reaction FEC codes to yield an overall encoding method for new enhanced systematic FEC codes. There is also a corresponding new decoding method that can be applied.

Decoder-in-the-Encoder Example

[0096] An example of a decoder in an encoder will now be provided.

[0097] Let encoding method E be an encoding method used by an encoder (in a transmitter or elsewhere) for a fixed-rate (non-systematic or systematic) FEC code E that generates N encoded symbols from K source symbols, where N is at least K. Similarly, let decoding method E be the corresponding decoding method for FEC code E, used by a decoder in a receiver or elsewhere.

[0098] Suppose FEC code E has the property that a random set of K out of the N encoded symbols are sufficient to recover the original K source symbols with reasonable probability using decoding method E, where reasonable probability might, for example, be probability 1/2. The reasonable probability can be some requirement set by the use or the application and might be a value other than 1/2. It should be understood that the construction of a particular code need not be specific to a particular recovery probability, but that applications and systems can be designed to their particular level of robustness. In some instances, the recovery probability can be increased by considering more than K symbols, and then determining using a decoding process a set of K symbols out of these considered symbols that allows successful decoding.

[0099] Suppose that for FEC code E, an ESI (Encoded Symbol Identifier) is associated with each encoded symbol and that ESI identifies that encoded symbol. Without loss of generality, the ESIs are labeled herein with 0, 1, 2, …, N-1.

[0100] In one embodiment of a systematic encoding method F for a systematic FEC code F generated using the methods for FEC code E, K and N are input parameters. The source symbols for FEC code F will have ESIs 0, …, K-1 and the repair symbols for FEC code F will have ESIs K, …, N-1. The systematic encoding method F for FEC code F generates N encoded symbols from K source symbols C(0), …, C(K-1) using encoding method E and decoding method E for FEC code E, performed by hardware and/or software as follows:

[0101] (1) randomly permute the N ESIs associated with FEC code E to arrive at the FEC code E permuted ESI set X(0), …, X(N-1), wherein this permuted ESI set is organized in such a way that the K source symbols of FEC code E can be decoded from the first K encoded symbols of FEC code E with respect to the permutation order of ESIs X(0), …, X(K-1),

[0102] (2) for each i = 0, …, N-1, associate ESI i of FEC code F with ESI X(i) of FEC code E,

[0103] (3) for each i = 0, …, K-1, set the value of the FEC code E encoded symbol with ESI X(i) to the value of source symbol C(i),

[0104] (4) apply the decoding method E to the source symbols C(0), …, C(K-1) with corresponding FEC code E ESIs X(0), …, X(K-1) to generate the decoded symbols E(0), …, E(K-1), and

[0105] (5) apply the encoding method E to the decoded symbols E(0), …, E(K-1) to generate FEC code E encoded symbols D(0), …, D(N-1) with associated FEC code ESIs 0, …, N-1,

[0106] (6) the encoded symbols for encoding method F with ESIs 0, 1, …, N-1 are D(X(0)), D(X(1)), …, D(X(N-1)).

[0107] Note that the output of encoding method F is N encoded symbols, of which the first K are the source symbols C(0), …, C(K-1) with associated ESIs 0, 1, …, K-1. Thus, encoding method F produces a systematic encoding of the source data.

[0108] One embodiment of a decoding method F that corresponds to the encoding method F just described is the following, where K and N are input parameters to this method that are used throughout. This decoding method F recovers K source symbols C(0), …, C(K-1) from K received encoded symbols D(0), …, D(K-1) with associated FEC code F ESIs Y(0), …, Y(K-1). The received symbols need not be exactly the sent symbols. The method, performed by hardware and/or software, is as follows:

[0109] (1) randomly permute the N ESIs associated with FEC code E to arrive at the FEC code E permuted ESI set X(0), …, X(N-1), wherein this permuted ESI set is organized in such a way that the K source symbols of FEC code E can be decoded from the first K encoded symbols of FEC code E with respect to the permutation order of ESIs X(0), …, X(K-1),

[0110] (2) apply decoding method E to the encoded symbols D(0), …, D(K-1) with associated FEC code E ESIs X(Y(0)), …, X(Y(K-1)) to generate decoded symbols E(0), …, E(K-1),

[0111] (3) using encoding method E, generate the encoded symbols C(0), …, C(K-1) with FEC code E ESIs X(0), …, X(K-1) from E(0), …, E(K-1),

[0112] (4) the decoded source symbols of FEC code F with ESIs 0, …, K-1 are C(0), …, C(K-1).

[0113] Methods and apparatus that operate as just described have some desirable properties. For example, consider an FEC code E that is a systematic code and has the property that a random set of K received encoded symbols can be decoded with high probability, but also has the property that when K encoded symbols are received and the proportion of source symbols among the received encoded symbols is not close to K/N, then it cannot be decoded with high probability. In this case, the embodiment describes a new FEC code F that uses the encoding and decoding methods of FEC code E, and the new FEC code F has the desirable property that it will decode with high probability from a set of K received encoded symbols, independent of the proportion of the received encoded symbols that are source symbols.

[0114] There are many variants of the above embodiment. For example, in step (1) of the encoding method F, the random permutation of the ESIs could be pseudorandom or based on some other method that produces a good selection of the ESIs but is neither random nor pseudorandom. In the case that FEC code E is a systematic code, it is preferable that the fraction of the first K ESIs in the permutation selected in step (1) from among the systematic ESIs is proportional to the rate of FEC code E, i.e., proportional to K/N. It is preferable that the random choices of the ESIs made by new encoding method F in step (1) can be represented by a succinct amount of data, for example by a seed to a well-known or agreed upon pseudorandom generator together with a agreed upon method to choose the ESIs based on the seed and how the pseudorandom generator works, so that the new decoding method F can make exactly the same ESI permutation choice in step (1) based on the same seed and pseudorandom generator and methods for generating ESIs. In general, it is preferable if the process used in new encoding method F in step (1) to generate the sequence of ESIs and the process used in new decoding method F in step (1) to generate the sequence of ESIs both generate the same sequence of ESIs, to ensure that new decoding method F is the inverse of new encoding method F.

[0115] There are other variants as well, where for example explicit ESIs are not used, but instead the unique identifier of an encoded symbol is by its position with respect to other encoded symbols, or by other means.

[0116] In the description above, the original ESIs of the FEC code E are remapped by the FEC code F so that the ordered set of source symbols are assigned the ESIs 0, …, K-1 in consecutive order, and the repair symbols are assigned the ESIs K, …, N-1. Other variants are possible, for example the remapping of ESIs can occur at a sender just after encoding method F has generated the encoded symbols but before the encoded symbols are transmitted, and the inverse remapping of ESIs can occur at a receiver as the encoded symbols are received but before the encoded symbols are processed for recovery of the original source symbols by decoding method F.

[0117] As another variant, in step (1) of new encoding method F the permutation might be selected by first selecting K+A FEC code E ESIs, where A is a value that is chosen to ensure decodability with high probability, and then during a simulation of the decoding process it is determined which of the K out of K+A ESIs are actually used during decoding, and the permutation selected might select the K ESIs actually used during decoding out of the initial set of K+A ESIs to be the first K ESIs of the permutation. Similar variants apply to new decoding method F.

[0118] As another variant of encoding method F, a seed that is used to generate the random permutation is pre-computed for a value of K to ensure that the first K encoded symbols of FEC code E associated with the permutation of ESIs produced in step (1) is decodable, and then this seed is always used for K in step (1) of encoding method F and corresponding decoding method F to generate the permutation in step (1). Methods for choosing such a seed include randomly choosing seeds until one is found that ensures decodability in step (1) and then selecting this seed. Alternatively, the seed could be dynamically generated with these properties by encoding method F and then this seed could be communicated to decoding method F.

[0119] As another variant of encoding method F, a partial permutation might be selected in step (1), i.e., not all of the ESIs need be generated in step (1) of new encoding method F, and not all of the encoded symbols need be generated if they are not needed in steps (5) and (6), e.g., because they correspond to source symbols that are part of the encoded symbols, or because less than N encoded symbols need to be generated. In other variants, not all of the encoded symbols in steps (3) and (4) of new decoding method F need be recomputed, as some of the received encoded symbols may correspond to some of the source symbols that are being recovered. Similarly, in step (2) of new decoding method F, not all K symbols E(0), …, E(K-1) need be decoded, for example if some of the symbols decoded in step (2) are not needed in subsequent steps to generate encoded symbols.

[0120] The methods and embodiments described above have many applications. For example, encoding method F and decoding method F and their variants can be applied to Tornado codes and to IETF LDPC codes to provide improved reception overhead and decoding failure probability performance. In general, these new methods apply to any fixed rate FEC code. Variants of these new methods can also be applied to FEC codes that have no fixed rate, i.e., to FEC codes such as chain reaction codes where the number of encoded symbols that can be generated is independent of the number of source symbols.

[0121] Shokrollahi III contains similar teachings for creating systematic encoding and decoding methods for chain reaction codes. In some embodiments, the encoding and the decoding methods E used for these codes are those taught in Luby I, Luby II, Shokrollahi I, Shokrollahi II, Luby III, Shokrollahi IV. To describe systematic encoders, it is often sufficient to describe encoding method E and decoding method E and use the general principles described above and known from those references to transform these methods to systematic encoding methods F and systematic decoding methods F. It should thus be apparent to one of ordinary skill in the art, upon reading this disclosure and the cited references, how to take the teachings that describe the encoding methods E and the decoding methods E and to apply the same to systematic encoding methods F and systematic decoding methods F, or the like.

Inactivation

[0122] Inactivation decoding, as taught in Shokrollahi II, is a general method that can be applied in combination with belief propagation whenever solving for a set of unknown variables from a set of known linear equation values, and is particularly beneficial when implementing efficient encoding and decoding methods that are based on sets of linear equations. In order to distinguish between inactivation decoding as described in Shokrollahi

II and permanent inactivation decoding as described herein below, “on the fly” inactivation (abbreviated to “OTF inactivation” in places) is used to refer to the methods and teachings of Shokrollahi II, whereas “permanent inactivation” is used to refer to the methods and teachings herein where inactivations are selected in advance.

[0123] One tenet of belief propagation decoding is that, whenever possible during the decoding process, the decoder should use a (possibly reduced) equation that depends on one remaining unknown variable to solve for that variable, and that equation is thus associated with that variable, and then reduce remaining unused equations by eliminating the dependence of those equations on the solved variable. Such a simple belief-propagation based decoding process has been used, for example, in some of the embodiments of Tornado codes, the chain reaction codes as described in Luby I, Luby II, Shokrollahi I, Shokrollahi II, Luby III, Shokrollahi IV, and the IETF LDPC codes.

[0124] OTF inactivation decoding goes in multiple phases. In a first phase of an OTF inactivation decoding method, whenever the belief propagation decoding process cannot continue because there is no remaining equation that depends on just one remaining unknown variable, the decoder will “OTF inactivate” one or more unknown variables and consider them “solved” with respect to the belief propagation process and “eliminated” from the remaining equations (even though they really are not), thus possibly allowing the belief propagation decoding process to continue. The variables that are OTF inactivated during the first phase are then solved for, for example using Gaussian elimination or more computationally efficient methods, in for example a second phase, and then in a third phase, the values of these OTF inactivated variables are used to fully solve for the variable associated with the equations during the first phase of decoding.

[0125] OTF inactivation decoding, as taught in greater detail in Shokrollahi II, can be applied to many other types of codes beyond chain reaction codes. For example, it can be applied to the general class of LDPC and LDGM codes, in particular to the IETF LDPC codes and to the Tornado codes, resulting in improvements in reliability (decreasing the probability of failing to decode) and/or CPU and/or memory performance (increasing the speed of encoding and/or decoding and/or decreasing the memory size required and/or access pattern requirements) for those types of codes.

[0126] Some of the variants of chain reaction code embodiments in combination with OTF inactivation decoding are described in Shokrollahi IV. Other variants are described in the present application.

System Overview

[0127] Fig. 1 is a block diagram of a communications system 100 that uses multi-stage coding. It is similar to that shown in Shokrollahi I, but in this case the encoder 115 takes into account a designation of which intermediate symbols are "permanently inactivated" and operates differently on those intermediate symbols than the intermediate symbols that are not permanently inactivated during the dynamic encoding process. Likewise, the decoder 155 also takes the permanently inactivated intermediate symbols into account when decoding.

[0128] As illustrated in Fig. 1, K source symbols (C(0), …, C(K-1)) are input to encoder 115 and, if decoding is successful with the symbols that become available to decoder 155, then decoder 115 can output a copy of those K source symbols. In some embodiments, a stream is parsed into K-symbol blocks and in some embodiments, a file of some number of source symbols larger than K is divided into K-sized symbol blocks and so transmitted. In some embodiments, where a block size of K′ > K is preferred, K′ - K padding symbols can be added to the K source symbols. These padding symbols can have values 0, or any other fixed value that is known to both encoder 115 and decoder 155 (or is otherwise able to be determined at decoder 155). It should be understood that encoder 115 might comprise multiple encoders, modules or the like, and that may also be the case for decoder 155.

[0129] As illustrated, encoder 115 also receives a sequence of dynamic keys from a dynamic key generator 120 and a sequence of static keys from as static key generator 130, each of which might be driven by a random number generator 135. The output of dynamic key generator 120 might be simply a cardinal number sequence, but that need not be the case. The operation of the key generators might be as shown in Shokrollahi I.

[0130] It should be understood that various functional blocks shown in the figures can be implemented as hardware with the specified inputs provided as input signals, or they can be implemented by a processor executing instructions that are stored in an instruction memory and executed in the appropriate order to perform the corresponding function. In some cases, specialized hardware is used to perform the functions and/or execute program code. Program code and processor are not always shown, but one of ordinary skill would know how to implement such details upon reading this disclosure.

[0131] Encoder 115 also receives inputs from an inactivation designator 125 and other parameters input to system 100 along the lines described elsewhere herein. Outputs of inactivation designator 125 might include a value, P, representing the number of intermediate symbols that are designated as "permanently inactivated" for decoding purposes (the "PI list" indicates which P of the intermediate symbols are on the list). As explained elsewhere, the intermediate symbols used for encoding processes are just the K source symbols in some embodiments, but in other embodiments, there is some type of processing, conversion, encoding, decoding, etc. that generates the intermediate symbols from the K source symbols beyond just copying them.

[0132] Input parameters might include random seeds used by the key generators and/or the encoder's encoding processes (described in more detail below), the number of encoded symbols to generate, the number of LDPC symbols to generate, the number of HDPC symbols to generate, the number of intermediate symbols to generate, the number of redundant symbols to generate, etc. and/or some of these values are calculated from other values available to encoder 115. For example, the number of LDPC symbols to be generated might be calculated entirely from a fixed formula and the value of K.

[0133] Encoder 115 generates, from its inputs, a sequence of encoded symbols ($B(I_0)$, $B(I_1)$, $B(I_2)$, …) and provides them to a transmit module 140 that also receives the dynamic key values (I_0, I_1, I_2, …) from dynamic key generator 120, but this might not be necessary if there is another method of conveying that information. Transmit module 140 conveys what it is given to a channel 145, possibly in a conventional manner that does not need to be described here in more detail. A receive module 150 receives the encoded symbols and the dynamic key values (where needed). Channel 145 may be a channel through space (for transmitting from one place to be received at another place) or a channel through time (for recording to media, for example, for replay back at a later time). Channel 145 may cause the loss of some of the encoded symbols. Thus, the encoded symbols $B(I_a)$, $B(I_b)$, … that decoder 115 receives from receive module 150 might not equal the encoded symbols that transmit modules sent. This is indicated by the different subscripted indices.

[0134] Decoder 155 is preferably able to regenerate the keys used for the received symbols (which keys might differ), using dynamic key regenerator 160, random number generator 163 and static key generator 165, and to receive as inputs various decoding parameters. Some of these inputs might be hardcoded (i.e., input during construction of a device) and some might be changeable inputs.

[0135] Fig. 2 is a table of variables, arrays and the like, with a summary of the notation that is most often used in the other figures and throughout this disclosure. Unless stated otherwise, K denotes the number of source symbols for the encoder, R denotes the number of redundant symbols generated by a static encoder, and L is the number of "intermediate symbols," i.e., the combination of source and redundant symbols and so L = K+R.

[0136] As is explained below, in some embodiments of a static encoder, two types of redundant symbols are generated. In a specific embodiment, used in many examples here, the first set comprises LDPC symbols and the second set comprises HDPC symbols. Without loss of generality, many examples herein refer to S as the number of LDPC symbols and H as the number of HDPC symbols. There might be more than two types of redundant symbols, so it is not required that R = S+H. LDPC symbols and HDPC symbols have different degree distributions and a person of ordinary skill in the art, upon reading this disclosure, would see how to use redundant symbols that are not LDPC or HDPC symbols, but where the redundant symbols comprise two (or more) sets of symbols wherein each set has a degree distribution distinct from the degree distributions of the other sets. As is well known, the degree distribution of a set of redundant symbols refers to the distribution of degree, wherein the degree of a redundant symbol refers to the number of source symbols upon which the redundant symbol depends.

[0137] P denotes the number of permanently inactive symbols among the intermediate symbols. The permanently inactive symbols are those that are designated for a particular treatment, namely to be "set aside" or "inactivated" in a belief propagation network in order to continue the belief propagation (and then come back to solve after solving the inactivated symbols), wherein permanently inactivated symbols are distinguished from other inactivated symbols in that the permanently inactivated symbols are designated at the encoder for such treatment.

[0138] N denotes the number of received symbols on which a decoding attempt is made by decoder 155, and A is the number of "overhead" symbols, i.e., the number of received encoded symbols beyond K. Hence, A = N-K.

[0139] K, R, S, H, P, N and A are integers, typically all greater than or equal to one, but in specific embodiments, some of these can be one or zero (e.g., R=0 is the case where there are no redundant symbols and P=0 falls back to the case of Shokrollahi II, where there is only OTF inactivation.)

[0140] The vector of source symbols is denoted by (C(0), …, C(K-1)) and the vector of redundant symbols is denoted by (C(K), …, C(L-1)). Therefore, (C(0), …, C(L-1)) denotes the vector of intermediate symbols, in the systematic case. A number, P, of those intermediate symbols are designated "permanently inactive." A "PI list" indicates which ones of the intermediate symbols are the permanently inactive ones. In many embodiments, the PI list simply points to the last P intermediate symbols, i.e., C(L-P), …, C(L-1), but this is not a requirement. That case is assumed only to simplify the remaining portions of this description.

[0141] The intermediate symbols that are not on the PI list are referred to as "LT intermediate symbols" herein. In the example, the LT intermediate symbols would be C(0), …, C(L-P-1). D(0), …, D(N-1) denote the received encoded symbols.

[0142] It should be noted that where an array of values is described as "N(0), …, N(x)" or the like, it should not be assumed that this requires at least three values, as it is not intended to exclude the case where there is only one or two values.

Encoding method using permanent inactivation

[0143] Fig. 3 is a block diagram of one specific embodiment of encoder 115 shown in Fig. 1. As illustrated there, the source symbols are stored in an input buffer 205 and provided to a static encoder 210 and a dynamic encoder 220, which also receive key inputs and other inputs. Static encoder 210 might include internal storage 215 (memory, buffer, virtual memory, register storage, etc.) for storing internal values and program instructions. Likewise, dynamic encoder 220 might include internal storage 225 (memory, buffer, virtual memory, register storage, etc.) for storing internal values and program instructions.

[0144] In some embodiments, a redundancy calculator 230 determines the number R of redundant symbols to create. In some embodiments, static encoder 210 generates two distinct sets of redundant symbols and in a specific embodiment, the first set is the first S redundant symbols, i.e., symbols C(K), …, C(K+S-1) and they are LDPC symbols, while the second set is the next H redundant symbols, i.e., C(L-H), …, C(L-1) and they are HDPC symbols. If the PI list is the last P redundant symbols, then all of the H redundant symbols may be on the PI list (if $P \geq H$) or all of the P redundant symbols may be HDPC symbols (if $P < H$).

[0145] The operations leading to the generation of these two sets of symbols may be quite different. For example, in some embodiments described below, the operations for generating the LDPC redundant symbols are binary operations and the operations for generating the HDPC symbols are non-binary.

[0146] The operation of dynamic encoder 220 is explained in further detail in Fig. 4. According to one embodiment, dynamic encoder 220 comprises two encoders, a PI encoder 240 and an LT encoder 250. In some embodiments, LT encoder 250 is a chain reaction encoder and PI encoder 240 is a chain reaction encoder of a particular type. In other embodiments, these two encoders may be very similar, or PI encoder 240 is not a chain reaction encoder. No matter how these encoders are defined, they generate symbols, wherein LT encoder 250 generates its symbols from the LT intermediate symbols C(0), …, C(L-P-1) that are designated as not permanently inactive, and whereas PI encoder 240 generates its symbols from the permanently inactive intermediate symbols C(L-P), …, C(L-1). These two generated symbols enter combiner 260 that generates the final encoded symbol 270.

[0147] In some embodiments of the present invention some of the permanently inactivated symbols may participate in the LT-encoding process, and some of the symbols that are not permanently inactivated symbols may participate in the PI encoding process. In other words, the PI list and the set of symbols comprising the LT intermediate symbols need not be disjoint.

[0148] In preferred embodiments, the symbols provided to combiner 260 may have the same length, and the function performed by combiner 260 is an XOR operation on these symbols to generate the encoded symbol 270. This is, however, not necessary for the working of this invention. Other types of combiners can be envisioned that could lead to similar results.

[0149] In other embodiments, the intermediate symbols are subdivided into more than two sets, for example one set of LT symbols and several (more than one) sets of PI symbols, each with its associated encoder 240. Of course, each associated encoder might be implemented as a common computing element or hardware element that operates on different instructions according to an encoding process when acting as a different encoder for different sets.

[0150] An example operation of PI encoding process 241, as might be performed by PI encoder 240, is exemplified in Fig. 5. Using the key I_a corresponding to an encoded symbol to be generated, at step 261, the encoder determines a positive weight, WP, and a list, ALP, containing WP integers between L-P and L-1, inclusive. In step 263, if list ALP = (t(0), …, t(WP-1)), then the value of a symbol X is set to $X = C(t(0)) \oplus C(t(1)) \oplus \ldots \oplus C(t(WP-1))$, wherein $\oplus$ denotes the XOR operation.

[0151] In some embodiments, the weight WP is fixed to some number, such as 3, or 4 or some other fixed number. In other embodiments, the weight WP may belong to a small set of possible such numbers, such as being chosen to be equal to either 2 or 3. For example, as shown in the embodiment of Appendix A, the weight WP depends on the weight of the symbol generated by LT encoding process 251, as might be performed by LT encoder 250. If the weight generated by the LT encoder 250 is 2, then WP is chosen to be either 2 or 3, depending on the key I_a, wherein the proportion of times in which WP is 2 or 3 is roughly equal; if the weight generated by the LT encoder 250 is larger than 3, then WP is chosen to be 2.

[0152] Fig. 6 is an example of an LT encoding process 251 according to one of the embodiments of the present invention and using the teachings of Luby I and Shokrollahi I. In step 267, the key I_a is used to generate a weight, WL, and a list, AL, respectively. In step 269, if list ALP = (j(0), …, j(WL-1)), then the value of a symbol X is set to $X = C(j(0)) \oplus C(j(1)) \oplus \ldots \oplus C(j(WL-1))$.

[0153] Fig. 7 illustrates an operation of calculating the weight WL. As shown there, in step 272, a number, v, is created that is associated with the encoded symbol to be generated and may be computed based on the key I_a for that encoded symbol. It can be the index, the representative label, etc. of the encoded symbol, or a distinct number, so long as encoders and decoders can be consistent. In this example, v is between 0 and 2^{20}, but in other examples, other ranges are possible (such as 0 to 2^{32}). The generation of v can be done in an explicit way using randomness generating tables, but the exact operation of how to generate these random numbers can vary.

[0154] The encoder is assumed to have access to a table M, an example of which is provided in Fig. 8. Table M, called a "degree distribution lookup" table, contains two columns and multiple rows. The left column is labeled with possible values of the weight WL, and the right column is labeled with integers between 0 and 2^{20}, inclusive. For any value of v, there is exactly one cell in the $M[d]$ column of the degree distribution lookup table wherein $M[d\text{-}1] < v \leq M[d]$ is true. For that one cell, there is a corresponding value in the d column, and the encoder uses that as the weight WL for the encoded symbol. For example, where an encoded symbol has v = 900,000, the weight for that encoded symbol would be WL = 7.

[0155] Static encoder 210 has access to elements SE(k,j) where k = 0, …, R-1 and j = 0, …, L-1. These element can belong to any finite field for which there is an operation * between elements α of the field and symbols X such that α*X is a symbol, and α*(X⊕Y) = α*X ⊕ α*Y where ⊕ denotes the XOR operation. Such fields and operations have been detailed in Shokrollahi IV. The operation of static encoder 210 can be described as computing, for a given sequence of source symbols C(0), …, C(K-1), a sequence of redundant symbols C(K), …, C(L-1) satisfying the relation shown in Equation 1, wherein Z(0), …, Z(R-1) are values known to the encoder and the decoder (for example, 0).

$$\begin{pmatrix} SE(0,0) & SE(0,1) & \cdots & SE(0,L-2) & SE(0,L-1) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ SE(R-1,0) & SE(R-1,1) & \cdots & SE(R-1,L-2) & SE(R-1,L-1) \end{pmatrix} \bullet \begin{pmatrix} C(0) \\ \vdots \\ C(K-1) \\ C(K) \\ \vdots \\ C(L-1) \end{pmatrix} = \begin{pmatrix} Z(0) \\ \vdots \\ Z(R-1) \end{pmatrix}, \quad \text{(Equ. 1)}$$

[0156] In Equation 1, the entries SE(k,j) can all be binary, or some of them can belong to the field GF(2) while others belong to other fields. For example, the corresponding matrix of the embodiment of Appendix A is given in Fig. 9. It comprises two submatrices, one with S rows and one with H rows. The upper submatrix comprises two parts: the submatrix comprising the last P columns in which every row has two consecutive ones (where the positions are counted modulo P). The first W = L-P columns of this matrix comprise circulant matrices followed by an SxS identity matrix. The circulant matrices comprise B of the columns and each (except possibly the last) has S rows. The number of these circulant matrices is ceil(B/S). The columns in these circulant matrices have each exactly 3 ones. The first column of the k-th circulant matrix has ones at positions 0, (k+1) mod S, and (2k+1) mod S. The other columns are cyclic shifts of the first. The lower H rows in Fig. 9 comprise a matrix Q with entries in GF(256) followed by an HxH identity matrix.

[0157] If α denotes an element of GF(256) with minimal polynomial $x^8+x^4+x^3+x^2+1$, then the matrix Q is equal to the matrix given in Fig. 10. Here, $\Delta_1, \ldots, \Delta_{K+S-1}$ are columns of weight 2 for which the positions of the 2 nonzero entries are determined pseudorandomly according to the procedure outlined in Section 5.3.3.3. of Appendix A. For judicious choices of values S, P, and H (such as the ones provided in Appendix A), the matrix in Fig. 10 leads to excellent recovery properties of the corresponding code. The procedure described above is exemplified in Fig. 11. In step 276, the matrix SE is initialized to 0. In step 278, an input variable S, equal to the number of LDPC symbols, is provided to the process, and the values of SE(i,j) are set to 1 for pairs (i,j) such that i = j mod S, or i = (1+floor(j/S)) +j mod S, or i = 2*(1+floor(j/S))+j mod S. This step takes care of the circulant matrices in Fig. 9.

[0158] In step 280, the positions corresponding the identity matrix I_S in Fig. 9 are set to one. In step 282, the positions corresponding to the PI part of the matrix in Fig. 9 are set to 1. These positions are of the form (i,l) and (i,t), where l = i mod P and t = (i+1) mod P. In step 284, the positions corresponding to the matrix Q in Fig. 9 are set. Accordingly, the matrix Q is provided as an additional input to this step. In step 286, the positions corresponding to the identity matrix I_H in the matrix of Fig. 9 are set to one.

[0159] Other choices for matrix SE are possible and depend on the particular application and the requirements demanded of the overall code. No matter how the matrix in Equation 1 is chosen, the task of the static encoder 210 can be accomplished in a variety of ways. For example, Gaussian elimination can be used as a process to recover the unknown values C(K), …, C(L-1) as would be apparent to one of ordinary skill in the art upon reading this disclosure.

<u>Decoding and permanent inactivation</u>

[0160] The decoding problem can be stated as follows: decoder 155 has N encoded symbols $B(I_a)$, $B(I_b)$, … with corresponding keys I_a, I_b, … The entire set of these encoded symbols, or a subset thereof, may have been received by the decoder, whereas the other encoded symbols may have been given to the decoder by other means. The goal of the decoder is to recover the source symbols C(0), …, C(K-1). To simplify the presentation, we denote the received encoded symbols by D(0), …, D(N-1).

[0161] Many of the decoding operations can be succinctly described using the language of matrices and operations on such matrices, in particular solving systems of equations with such matrices. In the following description, equations can correspond to received encoded symbols and variables can correspond to the source symbols or a combined set of source and redundant symbols generated from the source symbols, often called intermediate symbols, that are to be solved for based on received encoded symbols. In the specification provided as Appendix A, the encoded symbols might be referred to as “encoding symbols” (and there are other variations), but it should be apparent after reading the entire specification and appendix how the references relate. It should also be understood that the matrices and operations and solutions to equations can be implemented as computer instructions corresponding to those mathematical operations, and indeed it is not practical to do such operations without a computer, processor, hardware or some electronic element.

[0162] Permanent inactivation is used to determine at the decoder a set of variables to inactivate, called the permanently inactivated symbols or variables, before the first phase of the decoding process is initiated. The permanent inactivation decoding methods described below can be applied either to existing codes, or codes can be specially designed to work even better in conjunction with permanent inactivation decoding. Permanent inactivation decoding methods can be applied to solving any system of linear equations, and in particular can be applied to chain reaction codes, IETF LDPC codes and Tornado codes.

[0163] Permanent inactivation decoding is a general method that can be applied in combination with belief propagation decoding and/or OTF inactivation decoding whenever solving for a set of unknown variables from a set of known linear equation values, and is particularly beneficial when implementing efficient encoding and decoding methods that are based on sets of linear equations. In a first phase, based on the structure of the known encoding method or based on the received equations, a set of unknown variables are declared to be permanently inactivated, and the permanently inactivated variables are removed from the linear equations and considered to be "solved" in the second phase of the decoding process (except that as the second phase linear equations are reduced, the same reductions are performed on the permanently inactivated variables).

[0164] In the second phase, either belief propagation decoding is applied to the unknown variables that are not permanently inactivated using belief propagation decoding described previously, or OTF inactivation decoding is applied to the unknown variables that are not permanently inactivated, similar to that described for first phase of the OTF inactivation decoding method, thereby producing a set of reduced encoded symbols or equations. The reduced encoded symbols or equations that result from the second phase have the property that their dependency on the variables or symbols that are not inactivated has been eliminated, and thus the reduced encoded symbols or equations depend only on the inactivated variables or symbols. Note that an original encoded symbols or equations may be kept as well, so that both the original encoded symbols and the reduced encoded symbols may be available in some implementations.

[0165] In a third phase, the permanently inactivated variables together with any additional OTF inactivated variables generated in the second phase using OTF inactivation decoding are solved for using the reduced encoded symbols or equations, for example using Gaussian elimination, or, if it exists, a special structure of the relationships between the permanently inactivated variables and the linear equations is used to solve more efficiently than by using Gaussian elimination.

[0166] In a fourth phase, the values of the solved for inactivated variables, either OTF inactivated variables or permanently inactivated variables, are used in conjunction with the original encoded symbols or equations (or rederived original encoded symbols or equations) to solve for the variables that were not inactivated.

[0167] One of the advantages of permanent inactivation decoding methods is that the number w of OTF inactivations other than the permanent inactivations can be generally small or zero and can be largely independent of which encoded symbols are received. This can make the decoding complexity consistently small independent of which encoded symbols are received, allow more reliable decoding, and allow more predictable and fewer memory accesses which can be more efficiently scheduled. Because there are only a small number of OTF inactivations in the second phase, and because OTF inactivations in the second phase are generally only determined during the decoding process which can make the pattern of symbol operations somewhat unpredictable, the memory access patterns are more predictable during decoding, overall allowing more predictable efficient decoding processes.

[0168] There are many variants of the above. For example, the phases may be executed in non-sequential interleaved order. As another example, the inactivated symbols may in turn be solved for in the third phase using either OTF inactivation decoding or permanent inactivation decoding in multiple additional phases. As another example, permanent inactivation decoding may be applied to a linear system of equations and variables that may be used for error-correcting codes, or erasure correcting codes, or for other applications that can be solved using linear systems of equations. As another example, these methods can be applied to both systematic codes and to non-systematic codes. As another example, these methods can also be applied during an encoding process, for example when encoding using the methods taught in Shokrollahi III for generating systematic codes from non-systematic codes.

[0169] In some cases, it is possible to design the encoding process so that permanent inactivation decoding methods will be especially effective. For example, belief propagation decoding is known to be computationally efficient whenever it can be applied, but it is also known that it cannot provide high reliability decoding when used alone. When belief propagation decoding is used within OTF inactivation decoding, the belief propagation steps can be processed very efficiently, but the OTF inactivation steps interspersed within the belief propagation steps can slow down decoding, and the more such OTF inactivation steps there are, the slower is the decoding process.

[0170] In typical embodiments of OTF inactivation decoding, when trying to solve for K+R unknown variables using N+R linear equation values, the number of OTF inactivation steps is typically largest when N = K, i.e., when trying to solve the variables using zero overhead. On the other hand, as N grows larger than K, it is typically the case that the complexity of OTF inactivation decoding decreases due to fewer OTF inactivation steps, until when N is large enough so there are in some cases no OTF inactivation steps and inactivation decoding is as, or nearly as, computationally efficient as belief propagation decoding. In other embodiments of OTF inactivation decoding, the number of OTF inactivations may remain large even when N is considerably larger than K.

[0171] In one preferred embodiment of permanent inactivation decoding, the number P of permanently inactivated variables and the structure of the linear equations is designed so that when solving for the L-P variables that are not permanently inactivated using OTF inactivation decoding from K+R values of linear equations, the number of OTF inactivation steps during OTF inactivation decoding is small and in some cases zero, and thus the OTF inactivation decoding step is nearly as computationally efficient as belief propagation.

[0172] In preferred embodiments, the structure of the linear equations is designed such that the OTF inactivation decoding phase is nearly as efficient as belief propagation decoding. In such preferred embodiments, the relationships of the permanently inactivated variables to the linear equations is such that the phase of solving for the inactivated variables, comprised of the permanently inactivate variables together with any OTF inactivated variables from the OTF inactivation decoding phase, can be performed efficiently. Furthermore, in preferred embodiments the structure of the permanently inactivated symbols is such that the phase of completing the solution of the variables that are not inactivated from the solved inactivated variables is computationally efficient.

<u>Decoding of chain reaction codes with permanent inactivation</u>

[0173] Fig. 12 illustrates a matrix representation of a set of variables to be solved using N received encoded symbols or equations and R known static symbols or equations by the decoder. The task of the decoder is to solve the system of linear equations given in this figure. Typically, the symbols/equations are represented by values stored in memory or storage accessible by the decoder and the matrix operations described below are implemented by instructions executable by the decoder.

[0174] The matrix shown in Fig. 12 comprises L = K + R columns and N+R rows. The LT submatrix represents the relationships between the N encoded symbols and the L-P LT symbols of the L intermediate symbols determined by LT encoding process 251. The PI submatrix represents the relationships between the N encoded symbols and the P PI symbols of the L intermediate symbols determined by PI encoding process 241. The matrix SE of Equation 1 represents the relations among the intermediate symbols determined by the static encoder 210. The decoder can determine these relationships based on the keys for received encoded symbols and from the code construction.

[0175] The system of linear equations of Fig. 12 is solved by row/column permutations of the above matrix using the OTF inactivation methods taught in Shokrollahi II to transform it into a form shown in Fig. 13. It comprises a lower triangular matrix LO 310, a number of columns comprising matrix 320 (called OTFI) corresponding to OTF inactivations, a matrix 330 PI corresponding to the set of permanently inactive intermediate symbols or a subset thereof, and a matrix 340 EL corresponding to encoded or static symbols not used in the triangularization process leading to matrix LO.

[0176] Fig. 14 is a block diagram describing elements that might perform a process leading to the matrix in Fig. 12. It comprises an LT matrix generator 347, a PI matrix generator 349, and a static matrix generator 350. Upon receipt of keys I_a, I_b, … LT matrix generator creates the matrix LT in Fig. 12, whereas PI matrix generator 349 creates the matrix PI of Fig. 12. The concatenation of these two matrices is forwarded to static matrix generator 350, which may take as additional hints static keys S_0, S_1, … The task of static matrix generator is the creation of matrix SE, and its output is the full matrix given in Fig. 12.

[0177] The operations of LT matrix generator 347 and PI matrix generator 349 are tightly coupled with the operations of LT encoder 250 and PI encoder 240 in Fig. 15, respectively. Operation of static matrix generator 350 is the re-creation of matrix SE of Equation 1 used for static encoding.

[0178] LT matrix generator 347, PI matrix generator 349, and static matrix generator will now be described in further detail with reference to operations they might perform.

[0179] Fig. 16 is a flowchart illustrating one embodiment 500 of a method employed by LT matrix generator 347. In step 505, LT matrix generator 347 initializes a matrix LT of format N x (L-P) to all zeros. Next, in step 510, the keys I_a, I_b, … are used to generate the weights WL(0), …, WL(N-1), and the lists AL(0), …, AL(N-1), respectively. Each of the lists AL(i) comprises WL(i) integers (j(0), …, j(WL(i)-1)) in the range 0, …, L-P-1. In step 515, these integers are used to set entries LT(i,j(0)), …, LT(i,j(WL(i)-1)) to 1. As explained above, matrix LT contributes to a system of equations for the unknowns (C(0), …, C(L-1)) in terms of the received symbols (D(0), …, D(N-1)).

[0180] As can be appreciated by those skilled in the art, the operation of LT matrix generator as described here is similar to the operation of LT encoding process 251 of Fig. 6.

[0181] Fig. 17 is a flowchart illustrating one embodiment 600 of a method employed by PI matrix generator 349. In step 610, PI matrix generator 349 initializes a matrix PI of format N x P to all zeros. Next, in step 615, the keys I_a, I_b, … are used to generate weights WP(0), …, WP(N-1), and the lists ALP(0), …, ALP(N-1), respectively. Each of the lists ALP(i) comprises WP(i) integers (j(0), …, j(WP(i)-1)) in the range 0, …, P-1. In step 620, these integers are used to set entries PI(i,j(0)), …, PI(i,j(WP(i)-1)) to 1. The operation of PI matrix generator is similar to the operation of PI encoding process 241 in Fig. 5.

[0182] As explained above, the matrices LT and PI contribute to a system of equations in the unknowns (C(0), …, C(L-1)) in terms of the received symbols (D(0), …, D(N-1)). The reason is the following: once the LT encoder chooses weight WL(i) and associate list AL(i) = (j(0), …, j(WL(i)-1)), and PI encoder chooses weight WP(i) and associate list ALP(i) = (t(0), …, t(WP(i)-1)), the corresponding encoded symbol D(i) is obtained as shown below. These equations, accumulated for all values of i between 0 and N-1, give rise to the desired system of equations represented in Equation 2.

$$D(i) = C(j(0)) \oplus \ldots \oplus C(j(WL(i)-1)) \oplus C(t(0)) \oplus \ldots \oplus C(t(WP(i)-1)) \qquad \text{(Equ. 2)}$$

[0183] The weights WL can be calculated using a procedure similar to the one given in Fig. 7. A person of ordinary skill in the art, upon reviewing this disclosure, would see how to extend this to the case where there are more than two encoders, each operating with a different degree distribution.

[0184] A slightly different flow diagram of a matrix generator is provided in Fig. 18. It comprises an LT matrix generator 710, a static matrix generator 715, and a PI matrix generator 720. Upon receipt of keys I_a, I_b, …, LT matrix generator 710 creates the matrix LT illustrated in Fig. 15, whereas static matrix generator 715 creates the matrix SE illustrated in Fig. 15, and may take the additional static keys S_0, S_1, ... as its further input. The concatenation of these two matrices is forwarded to PI matrix generator 720 that creates the matrix PI. Operation of LT matrix generator 710 may be exactly the same as the operation of LT matrix generator 347 detailed in Fig. 16. The operation of static matrix generator 715 may be different from the operation of the static matrix generator 350 in Fig. 14. Specifically, Fig. 19 details an exemplary embodiment of such an operation.

[0185] In step 725, the matrix SE is initialized to 0. In step 730, an input variable S, equal to the number of LDPC symbols, is provided to the process, and the values of SE(i,j) are set to 1 for pairs (i,j) when i = j mod S, i = (1+floor(j/S))+j mod S, or i = 2*(1+floor(j/S))+j mod S. In step 735, the positions corresponding the identity matrix IS in Fig. 9 are set to one. In step 740, the positions corresponding to a matrix T are provided as an additional input to this step. This matrix may have entries in multiple finite fields, and can be different for different applications. It may be chosen based on requirements demanded of the code.

[0186] Fig. 20 is a simplified flow diagram illustrating one embodiment of a method employed by PI matrix generator 720. In step 745, PI matrix generator 349 initializes a matrix PI of format (N+R) x P to all zeros. Next, in step 750, the keys I_a, I_b, ... are used to generate weights WP(0), ..., WP(N-1), and the lists ALP(0), ..., ALP(N-1), respectively. Each of the lists ALP(i) comprises WP(i) integers (j(0), ..., j(WP(i)-1)) in the range 0, ..., P-1. In step 755, these integers are used to set entries PI(i,j(0)), ..., PI(i,j(WP(i)-1)) to 1. The operation of PI matrix generator in Fig. 20 is similar to the operation of the PI matrix generator of Fig. 17 with the exception that this matrix generator creates a matrix with R more rows and is tightly coupled with the matrix in Fig. 15.

[0187] The system of equations in Fig. 12 or in Fig. 15 is typically sparse, i.e., the number of nonzero entries in the matrices involved is typically much smaller than half the possible entries. In such a case, the matrices might need not be stored directly, but an indication may be stored that helps in recreating every individual entry of these matrices. For example, for every one of the rows of the matrices LT or PI, a process may want to store the weight and the list of neighbors as computed in Figs. 5-6. Other methods are also possible and many of them have been explained herein or in disclosures incorporated by reference herein.

[0188] Once the matrix generator has created a system of equations in the form given by Fig. 12 or Fig. 15, the task of the decoder is to solve this system for the unknown values of C(0),

…, C(L-1). A number of different methods can be applied to achieve this goal, including but not limited to Gaussian elimination, or any of the methods described in Luby I, Luby II, Shokrollahi I, II, III, IV, or V.

[0189] A possible method for solving the system of equations in Fig. 12 or Fig. 15 is now outlined with reference to Figs. 21-26. A flow chart of an operation of the decoder according to some of the embodiments of the present invention is given in Fig. 21. In step 1305, the decoding matrix is created using some of the methods described earlier. In step 1310, this matrix is rearranged using row and column permutations. As was mentioned above, such a matrix may be obtained from either of the matrices in Fig. 12 or Fig. 15 by applying row and column permutations. Chain reaction decoding in combination with on-the-fly inactivation decoding of Shokrollahi II can be used to achieve this. There are thus permutations **pi** operating on the set {0, 1, …, L-1} and **tau** operating on the set {0, 1, …, N+R-1} such that the equation in Fig. 22 is satisfied.

[0190] Herein, w denotes the number of rows and columns of matrix LO in Fig. 13, i.e., the number of intermediate symbols that are neither permanently, nor OTF inactivated. In step 1315, the matrix LO of Fig. 13 is used to zero out all entries of matrix LO below the diagonal. In doing so, the set of symbols on the right of the equation in Fig. 23 needs to respect the same operations, so that the new right hand side of the system of equations is obtained by XORs of some of the D(tau(i)).

[0191] As illustrated in Fig. 24, after this operation, matrix 810 becomes an identity matrix, matrix EL in 840 will be untouched, and matrices OTFI and PI will be changed to OTFI-2 in 820 and PI-2 in 830, because the decoding process needs to XOR rows of these matrices together according to the operations that were necessary to reduce matrix LO to the identity matrix.

[0192] A next step of the decoding process may be step 1320, in which the rest of the remaining matrix below LO is eliminated to obtain a matrix of the form indicated in Fig. 25. Denoting the permuted and reduced values of the original symbols D(0), …, D(N_R-1) after this step by E(0), …, E(N+R-1), by u the number of rows of the matrix EL_2, and by g the number of columns of EL_2, the structure of the matrix in Fig. 25 results in a smaller system of u linear equations for the values of C(pi(L-g)), …, C(pi(L-1)) according to Equation 3.

$$(EL\_2)\bullet\begin{pmatrix} C(pi(L-g)) \\ \vdots \\ C(pi(L-1)) \end{pmatrix} = \begin{pmatrix} E(N+R-u) \\ \vdots \\ E(N+R-1) \end{pmatrix}. \qquad \text{Equ. 3}$$

[0193] A decoding process such as the one described in Fig. 21 may solve this system of equations in step 1330 by a variety of means, for example by using a Gaussian elimination process, or a combination of chain reaction coding and Gaussian elimination, or by another application of inactivation decoding, or by other means. The Gaussian elimination can be modified so as to separate computations in GF(2) from those in larger fields, such as GF(256), if the matrix EL has elements belonging to multiple fields, as was taught in Shokrollahi IV, for example.

[0194] If the system of equations in Equation 3 is not solvable using the processes employed by the decoder, then the decoder may apply countermeasures in step 1335. Such countermeasures could include flagging an error and stopping the process, or it could include requesting more encoded symbols, or it could stop the process and give back to the application using the decoder a list of intermediate symbols or source symbols that it has been able to recover so far. If the system is solvable, then the decoder may recover the values of inactivated intermediate symbols C(pi(L-g)), …, C(pi(L-1)). In some variants, it may be as well that some other intermediate symbols besides the inactivated intermediate symbols are recovered in step 1330.

[0195] Once the values of these symbols are recovered, the decoder proceeds to step 1340 that involves a back substitution. Recovering the values of C(pi(L-g)), …, C(pi(L-1)) results in a system of equations of the type given in Fig. 26. This system is easier to solve than a general system. For example, a decoder may use the process indicated in Fig. 23 to do so. The process of obtaining the first vector on the right hand side of Fig. 23 may be referred to as back-substitution, as it is the process of substituting the values of the known symbols into the system of equations. As can be seen by a person of ordinary skill in the art after reading this disclosure, the systems given in Figs. 23 and 26 are mathematically equivalent.

[0196] In Fig. 23, the decoder obtains the unknown values C(pi(0)), …, C(pi(L-g-1)) by implementing a process in which the entries of the matrix on the right hand side are multiplied with the entries of the already solved for vector C(pi(L-g)), …, C(pi(L-1)) using the rules of matrix multiplication, and XORing the obtained entries with E(0), …, E(L-g-1).

The process of XORing the obtained entries with E(0), …, E(L-g-1) and thus recovering the values of C(pi(0)), …, C(pi(L-g-1)) comprises step 1345 of the decoder in Fig. 21.

[0197] Though useful in some applications, this method may lead to a large computational overhead in some preferred embodiments, since the matrix on the right hand side of Fig. 23 is typically not sparse and therefore, to obtain one of the elements C(pi(j)) a number of XORs has to be performed which is proportional to g. In some embodiments, this number may be large, for example because the number P of permanent inactivations was chosen to be large to begin with, and g may be at least as large as P. This can put severe limitations on the value of P, the number of permanently inactivated symbols, and if a smaller value of P is used, then this could lead to an increase in the number of OTF inactivated intermediate symbols.

[0198] Fig. 27 describes a modified decoding process that may be computationally more efficient than the process described in Fig. 21. Steps 1405 through 1435 of this process may be the same as the corresponding steps of the process in Fig. 14. Optionally, this process may keep a copy of the original matrix in Fig. 12 or Fig. 15, or relevant parts of this matrix, as well as the original symbols D(0), …, D(N+R-1) in an additional memory location for future use. This is not necessary for the working of this process, but it may lead to further speed advantages if the application has enough memory resources to keep these copies. Alternatively, the process may only keep a copy of the original symbols D(0), …, D(N+R-1) and not the matrix, and re-create the matrix when it needs it. Step 1440 either uses the stored copy of the matrix or undoes the process in step 1415 to obtain back the original system of equations in Fig. 22, or only the top part of this system given in Fig. 28. At this point, the matrix 1510 given in Fig. 29 is sparse, and the values C(pi(w)), …, C(pi(L-1)) are known, where w = L-g.

[0199] As is well known, the right hand side of the equation in Fig. 29 can be computed via a computationally efficient process involving a small number of XORs of symbols, i.e., equal to the number of non-zero entries in the matrix OTFI plus the number of non-zero entries in the matrix PI. This step of the process is denoted by 1445 in Fig. 27. After this step is complete, the right hand side of the equation in Fig. 29 has been computed, and a system of equations is to be solved in which the unknowns are the values of C(pi(0)), …, C(pi(w-1)). This system can be solved in step 1450 using chain reaction decoding, since the lower triangular LO on the right hand side is sparse, i.e., the number of XORs of symbols to solve this system of equations is equal to the number of non-zero entries in the matrix LO and this number is typically much smaller than w*w, the maximum number of non-zero entries possible.

Choice of the Number of Permanent Inactivations

[0200] The choice of the number of permanent inactivations can affect overall performance, so it can be important. On the one hand, this number needs to be chosen to be as large as possible: if this number is large, then the number of OTF inactivations may be reduced to a very small number, sometimes even zero. This is because the combination of the LT and the SE matrix in Fig. 15 (or their corresponding variants in Fig. 23) is effectively the decoding matrix of a chain reaction code with a large overhead. This fact makes the number of OTF inactivations very small. OTF inactivations may be harder to manage in certain embodiments, hence reducing their number may lead to advantages in terms of speed and/or memory.

[0201] On the other hand, increasing the number of permanent inactivations may have an adverse effect on the running time: for example, step 1330 in the decoding process of Fig. 21, and the corresponding step 1430 in the process of Fig. 27 require solving a system of equations that has at least P rows and columns. One way to do this would be to identify an invertible submatrix of the matrix EL-2 in Fig. 25, invert that matrix, and use the inverted matrix to obtain the values of the intermediate symbols C(pi(L-g-1)), …, C(pi(L-1)). Since the matrix EL-2 may not be sparse in many of the embodiments, obtaining the values of the intermediate symbols may incur on the order of g times g XORs of symbols. Since g is at least P, the number of XORs of symbols maybe at least P times P, so if the overall number of XORs of symbols is to be kept linear in K, a good choice is to set the number P to be proportional to the square root of K. The specific embodiment of Appendix A chooses P to be of the order of 2.5*sqrt(K), and keeps in line with this observation. This is a good choice for P, as with this choice of P, typically the number of OTF inactivations is fairly small, varying from around P to very close to or equal to zero.

[0202] Another quantity of interest is the average number, I, of inactivated intermediate symbol neighbors there are for an encoded symbol, or for a static symbol. Step 1445 of the decoding process in Fig. 27 may need as many as I XORs of symbols on average per unrecovered intermediate symbols to accomplish this step. If I is large, then this number of XORs may be too many for the memory and computational resources of the processes executing the decoding or the encoding process. On the other hand, if I is too small, then the matrix EL-2 of Fig. 25 may not have full rank, and decodability may be jeopardized.

[0203] A more detailed analysis reveals that an important aspect of permanent inactivation is to make the matrix PI of Fig. 15 behave in such a way that the columns are linearly independent of one another, i.e., the matrix is full rank as much as is possible. It is well-known to those of skill in the art that if PI is a random binary matrix, then full rank to the limits possible may be achieved. On the other hand, PI may have on average in each column a fraction of ones that is inversely proportional to the square root of K and still satisfy the same rank properties as that of a purely random matrix. For this reason, the specific embodiment in Appendix A chooses I to be a number between 2 and 3, and thus with the choice of P proportional to the square root of K, this means that the number of ones in each column of PI is on average inversely proportional to square root of K.

[0204] There are many variants of these methods, as one skilled in the art will recognize upon reading this disclosure. For example, XOR may be replaced with other operators, e.g., linear operators over larger finite fields, or the operators may be a mixture of different operators, e.g., some linear operators over larger finite fields for some of the operations and other linear operators over smaller larger finite fields for others of the operations.

Specific Example with reference to Appendix A

[0205] As detailed above, without permanent inactivations (i.e., predetermined decisions as to which encoded symbols would not be part of a matrix manipulation that would be part of determining a sequence for a chain reaction decoding), the number of OTF inactivations might be quite random and cause potential problems in terms of memory consumption. Where the number of source symbols is very large and the overhead is very small, the error probability can be unacceptably close to 1.

[0206] Because of the high error probability for small overheads, it can become increasingly difficult to find good systematic information when the number of source symbols is large. Herein, systematic information refers to information needed to provide to the encoder and decoder in order to be able to construct a systematic code in the sense of Shokrollahi III. Moreover, whenever systematic information is obtained, it is to be expected that the behavior of the code is very far from its average behavior, because on “average” the code should fail at zero overhead.

[0207] Some of the parameters for the construction of a chain reaction code with permanent inactivation may include the degree distribution Ω used for the LT encoder 250 of Fig. 4, the parameters for the PI encoder 240, the determination of the number of permanently inactivated symbol, the determination of the number of redundant static symbols and their structure, and the particular way random numbers may be generated and shared between encoder 115 and decoder 155 in Fig. 1.

Encoders and Decoders that use the RQ Code

[0208] A preferred embodiment of a code, hereafter referred to as "the RQ code", that uses the methods described herein is specified in great detail in Section 5 of Appendix A. The remainder of Appendix A describes one method of applying the RQ code to the reliable delivery of objects over broadcast or multicast networks.

[0209] The RQ code uses the methods described previously and below to implement a systematic code, meaning that all the source symbols are among the encoded symbols that can be generated, and thus encoded symbols can be considered to be a combination of the original source symbols and repair symbols generated by the encoder.

[0210] Although some of the previous codes have good properties, there are some improvements that would increase their practical application. Two potential improvements of importance are a steeper overhead-failure curve and a larger number of supported source symbols per source block. The overhead is the difference between the number of encoded symbols received and the number of source symbols in the source block, e.g., an overhead of 2 means that K+2 encoded symbols are received to decode a source block with K source symbols. The failure probability at a given overhead is the probability that the decoder fails to completely recover the source block when the number of received encoded symbols corresponds to that overhead. The overhead-failure curve is a plot of how the failure probability drops as a function of increasing overhead, starting at overhead zero. An overhead-failure curve is better if the failure probability of the decoder drops off fast, or steeply, as a function of overhead.

[0211] A random binary code has an overhead-failure probability curve where the failure probability drops by essentially a factor of two for each additonal overhead symbol, with unworkable computational complexity, but the subject of the current discussion is limited to the overhead-failure probability curve, and not computational complexity). In some applications, this is a sufficient overhead-failure curve, but for some other applications, a steeper overhead-failure curve is preferred. For example, in a streaming application, the range of the number of source symbols in a source block can be wide, e.g., K = 40, K = 200, K = 1,000, K = 10,000. To provide a good streaming experience the failure probability may be required to be low, e.g., a failure probability of 10^{-5} or 10^{-6}. Since bandwidth is often at a premium for streaming applications, the percentage of repair symbols sent as a fraction of the source symbols should be minimized. Suppose, for example, that the network over which the stream is sent should be protected against up to 10% packet loss when using source blocks with K = 200, and the failure probability is required to be at most 10^{-6}. A random binary code requires an overhead of at least 20 to achieve a failure probability of 10^{-6}, i.e., the receiver needs 220 encoded symbols to decode with this failure probability. A total of 245 encoded symbols need to be sent for each source block to meet the requirements, since ceil(220/(1-0.1)) = 245. Thus, the repair symbols add an extra 22.5% to the bandwidth requirements for the stream.

[0212] The RQ code described herein and in Section 5 of Appendix A achieves a failure probability that is smaller than 10^{-2}, 10^{-4}, and 10^{-6} for overheads 0, 1, and 2, respectively, for values of K = K′ for all supported values of K′ and for values of K = 1 and K = K′+1 for all but the final supported value of K. Tests have been done for a variety of loss probabilties, e.g., loss probabilities of 10%, 20%, 50%, 70%, 90% and 95%.

[0213] For the example above using the RQ code, an overhead of 2 is sufficient to achieve a failure probability of 10^{-6}, and thus only a total of 225 encoded symbols need to be sent for each source block to meet the requirements, since ceil(202/(1-0.1)) = 225. In this case, the repair symbols add an extra 12.5% to the bandwidth requirements for the stream, i.e., 10% less bandwidth overhead than required by a random binary code. Thus, the RQ code improved overhead-failure curve has some very positive practical consequences.

[0214] There are applications where support for a large number of source symbols per source block is desirable. For example, in a mobile file broadcast application, it is advantageous from a network efficiency point of view to encode the file as a single source block or, more generally, to partition the file into as few source blocks as is practical. Suppose for example that a file of 50 million bytes is to be broadcast, and that the available size within each packet for carrying an encoded symbol is one thousand bytes. To encode the file as a single source block requires that a value of K = 50,000 be supported. (Note that there are sub-blocking techniques as described previously that allow decoding using substantially less memory).

[0215] There are a few reasons that the number of source symbols supported for a code might be limited. One typical reason is that computational complexity becomes unreasonable as K increases, such as for Reed-Solomon codes, but this is not the case for codes such as chain reaction codes. Another reason might be that the failure probability at zero overhead increases to almost 1 as K increases, making it harder to find systematic indices that yield a good systematic code construction. The failure probability at zero overhead can dictate the difficulty of deriving a good code construction, because this essentially the probability that when a systematic index is chosen randomly that the resulting systematic code construction has the property that the first K encoded symbols are able to decode the K source symbols.

[0216] Because the overhead-failure curve for the the RQ code design is so steep for all values of K, it is easily possible to find good systematic indices and thus to support much larger values of K. The RQ code as described in Section 5 of Appendix A supports values of K up to 56,403, and also supports a total number of encoded symbols up to 16,777,216 per source block. These limits on supported values for the RQ code were set due to practical considerations based on perceived application requirements, and not due to limitations of the RQ code design. Other embodiments beyond those shown in Appendix A might have different values.

[0217] The RQ code limits the number of different source block sizes that are supported as follows. Given a source block with K source symbols to be encoded or decoded, a K′ value is selected based on the table shown in Section 5.6 of Appendix A. The first column in the table lists the possible values for K′. The value of K′ selected is the smallest value among the possibilities such that $K \leq K'$. The K source symbols C′(0), …, C′(K-1) are padded with K′-K symbols C′(K), …, C′(K′-1) with values set to zeroes to produce a source block comprising K′ source symbols C′(0), …, C′(K′-1), and then encoding and decoding are performed on this padded source block.

[0218] The above approach has the benefit of reducing the number of systematic indices that need to be supported, i.e., only a few hundred instead of tens of thousands. There is no disadvantage in terms of the overhead-failure probability for K, as it is the same as the overhead-failure curve for the selected K′: Given the value of K, the decoder can compute the value of K′, and set the values of C′(K), …, C′(K′-1) to zeroes, and thus it only has to decode the remaining K of the K′ source symbols of the source block. The only potential disadvantages are that slightly more memory or computational resources might be needed for encoding and decoding with slightly more source symbols. However, the spacing between consecutive values of K′ is roughly 1% for larger values of K′, and thus the potential disadvantage is negligible.

[0219] Because of the padding of the source block from K to K′, the identifier for encoded symbols C′(0), C′(1), ... within the RQ code is called the Internal Symbol Identifier, abbreviated to ISI, where C′(0), ..., C′(K′-1) are the source symbols and C′(K′), C′(K′+1), ... are the repair symbols.

[0220] External applications employing the encoder and decoder use an Encoded Symbol Identifier, also called an Encoding Symbol Identifier, abbreviated to ESI, that ranges from 0 to K-1 to identify the original source symbols C′(0), ..., C′(K-1) and that continues K, K+1, ... to identify repair symbols C′(K′), C′(K′+1), ... Thus, a repair symbol C′(X) identified with ISI X within the RQ code is identified externally with an ESI X-(K′-K). This is described in more detail in Section 5.3.1 of Appendix A.

[0221] The encoding and decoding for the RQ codes is defined by two types of relationships: constraint relationships among the intermediate symbols and LT-PI relationships between the intermediate symbols and the encoded symbols. The constraint relationships correspond to the relationships among the intermediate symbols defined by the SE matrix as for example shown in Fig. 12 or Fig. 15. The LT-PI relationships correspond to the relationships between the intermediate symbols and the encoded symbols defined by the LT matrix and PI matrix as for example shown in Fig. 12 or Fig. 15.

[0222] Encoding proceeds by determining the intermediate symbol values based on: (1) the source symbol values; (2) LT-PI relationships between the source source symbols and the intermediate symbols; and (3) the constraint relationships among the intermediate symbols. The values of repair symbols can be generated from the intermediate symbols based on LT-PI relationships between the intermediate symbols and the repair symbols.

[0223] Similarly, decoding proceeds by determining the intermediate symbol values based on: (1) the received encoded symbol values; (2) LT-PI relationships between the received encoded symbols and the intermediate symbols; and (3) the constraint relationships among the intermediate symbols. The values of missing source symbols can be generated from the intermediate symbols based on LT-PI relationships between the intermediate symbols and the missing source symbols. Thus, encoding and decoding are essentially symmetric procedures.

Example Hardware Components

[0224] Figs. 30-31 illustrate block diagrams of hardware that might be used to implement methods described above. Each element can be hardware, program code or instructions executed by a general purpose or custom-purpose processor or a combination.

[0225] Fig. 30 illustrates an example encoding system 1000, that might be implemented as hardware modules, software modules, or portions of program code stored in a program store 1002 and executed by a processor 1004, possibly as a collective unit of code not separated as shown in the figure. Encoding system 1000 receives a signal in, conveying source symbols and parameter information, and outputs a signal conveying that information.

[0226] An input interface 1006 stores the incoming source symbols into a source symbol buffer 1008. A source-to-intermediate symbol generator 1010 generates intermediate symbols from the source symbols. This can be a pass-through in some embodiments and a decoder module in other embodiments (such as a "systematic" embodiment).

[0227] A redundant symbol generator 1012 generates redundant symbols from the source symbols. This can be implemented as a chain reaction coder, an LDPC coder, an HDPC coder, or similar. An inactivator 1014 receives the source symbols, intermediate symbols and/or redundent symbols, as the case may be, and stores some of them, the permanently inactivated symbols, in a PI buffer 1018 and provides the others to an output encoder 1016. This process might only be logically, rather than physically.

[0228] An operator 1020, such as an XOR operator, operates on one or more encoded symbols from output encoder 1016 (one, in certain embodiments) and one or more of the PI symbols from PI buffer 1018 (one, in certain embodiments), and the result of the operation is provided to a transmit interface 1030 that outputs the signal from system 1000.

[0229] Fig. 31 illustrates an example decoding system 1100, that might be implemented as hardware modules, software modules, or portions of program code stored in a program store 1102 and executed by a processor 1104, possibly as a collective unit of code not separated as shown in the figure. Some process might only be logically, rather than physically, implemented.

[0230] Decoding system 1100 takes in an input signal and possibly other information and outputs souce data, if it is able to. The signal in is provided to a receive interface 1106 that stores received symbols in a buffer 1108. The ESIs of received symbols is provided to a matrix generator 1110 that generates matrixes as described herein, in dependence on the particular symbols received, and stores the results in a matrix memory 1112.

[0231] A scheduler 1114 can read matrix details from matrix memory 1112 and generates a schedule, stored in a schedule memory 1016. Schedule 1114 might also generate a done signal and convey a PI matrix to a PI solver 1118 when complete. PI solver 1118 provides solved PI symbol values to a solver 1120, which also used the schedule, to decode the intermediate symbols from the received symbols, schedule and PI symbols.

[0232] The intermediate symbols are provided to an intermediate-to-source symbol generator 1122, which could be an encoder or pass-through. The output of intermediate-to-source symbol generator 1122 is provided to an output interface 1124 that outputs the source data, or what source data is available for output.

Other considerations

[0233] In certain situations, there might be a need for enhanced decodability. In examples provided elsewhere herein, while encoded symbols had both LT neighbors and PI neighbors, the LDPC symbols only had LT neighbors or PI neighbors that were not among the HDPC symbols. In some instances, decodability is improved if LDPC symbols also have PI neighbors that include the HDPC symbols. With neighbors among all of the PI symbols, including the HDPC symbols, the decoding worth of the LDPC symbols might be more similar to that of the encoded symbols. As explained elsewhere herein, symbols that depend on the LT symbols (which can be easy to encode and decode) and also depend on the PI symbols, including the HDPC symbols (which can provide high reliability decoding), so that both advantages might be present.

[0234] In an example, each LDPC symbol has two PI neighbors, i.e., an LDPC symbol's value depends on the values of two PI symbols.

[0235] Decodability might also be improved, in some situations, reducing the occurrences of duplicate encoded symbols, where two encoded symbols are duplicates if they have exactly the same overall neighbor set, where the overall neighbor set for an encoded symbol is comprised of the LT neighbor set and the PI neighbor set. Duplicate encoded symbols with the same overall neighbor set carry exactly the same information about the intermediate source block from which they were generated, and thus there is no better chance at decoding from having received more than one duplicate encoded symbols than there is from having received one of the duplicate encoded symbols, i.e., reception of more than one duplicate symbol adds to the reception overhead and only one of the encoded symbols among the duplicates is useful for decoding.

[0236] A preferable property is that each received encoded symbol is not a duplicate of any other received encoded symbol, since this means that each received encoded symbol may be useful for decoding. Thus, it might be preferred to reduce the number of such duplications or reduce the probability of occurrence of duplicates.

[0237] One approach is to limit the number of LT neighbors that each encoded symbol can have. For example, if there are W possible neighbors, the maximum number of neighbors might be limited to W-2. This reduces the chance that overall neighborhood sets would be duplicated, in some cases, because the neighborhood set comprising all W possible neighbors would not be allowed. Where the constraint is Deg[v] = min(d, W-2), there are W*(W-1)/2 different neighborhood sets of degree W-2. Thus, it can be less likely that duplicate overall neighbor sets are generated for encoded symbols. Other constraints, such as min(d, W-Wg) for some Wg other than Wg = 2, or some other constraint, might be used instead.

[0238] Another technique, which can be used alone or with the above duplicate-reducing technique, is to choose more than one PI neighbor for each encoded symbol, so that it is less likely that there are duplicate PI neighbors for encoded symbols, and thus less likely that duplicate overall neighbor sets are generated for encoded symbols. The PI neighbors can be generated in similar ways to how the LT neighbors are generated, for example by first generating a (d1, a1, b1) as shown in the Appendix A, Section 5.3.5.4 according to the code snippet below:

```
if (d < 4) then {d1 = 2 + Rand[y, 3, 2]} else {d1 = 2};
a1 = 1 + Rand[y, 4, P1-1];
b1 = Rand[y, 5, P1];
```

[0239] Note that in this example, there is a non-trivial random degree distribution defined on the number of PI neighbors d1 and that distribution depends on the chosen number of LT neighbors d, and the number of PI neighbors is likely to be greater when the number of LT neighbors is smaller. This provides the property that the overall degree of the encoded symbol is such that it reduces the chance that duplicate encoded symbols will be generated and thus received.

[0240] The encoded symbol value might be generated using the neighbors defined by (d1, a1, b1) as shown in Appendix A, Section 5.3.5.3, and by the following code snippet:

```
while (b1 >= P) do {b1 = (b1+a1) % P1};
      result = result ^ C[W + b1];
For j = 1, ..., d1-1 do
      b1 = (b1+a1) % P1;
      while (b1 >= P) do {b1 = (b1+a1) % P1};
      result = result ^ C[W + b1];
      Return result;
```

[0241] To support these decodability features or separately to provide for decodability, a different systematic index J(K′) for values of K′ might be used, such as the one shown in Table 2 of Section 5.6 in Appendix A.

[0242] An example of a process that is performable in a transmission and/or reception system to generate systematic index J(K′) is illustrated as follows. For each K′ in the list of possible K′, one process that could be performed, typically by an appropriately programmed circuit or processor, is to check a number of indices for suitability. For example, the circuit/processor might check, for J = 1 ... 1000 [or some other limit], whether the following criteria are met with respect to possible systematic index J:

[0243] (a) Is decoding possible at zero overhead from the K′ source symbols?

[0244] If Yes, record the number of on-the-fly inactivations

[0245] (b) Are there duplicate overall neighbor sets among the first K′/0.06 possible encoded symbols (with ESIs 0, …, K′/0.06)? [Other thresholds might be used instead.]

[0246] (c) Is the decode failure probability below 0.007 [or some other threshold] when decoding using the first K′ received encoded symbols within 10,000 runs [or some other test] when each encoded symbol is lost with probability 0.93 [or some other threshold] in each run independently of the other encoded symbols?

[0247] The circuit/processor then chooses among the possible systematic indices J that satisfy criteria (a), (b) and (c) above, choosing the systematic index that recorded an average number of on-the-fly inactivations in step (a).

[0248] Note that there are many variations of the above selection criteria. For example, in some cases it might be preferable to choose the systematic index that satisfies (a), (b) and (c) above and yields the fewest number of decode failures in step (c) within the specified number of runs. As another example, a combination of the number of on-the-fly inactivations and the decode failure probability might be taken into consideration when choosing a systematic index. As another example, multiple systematic indices for each K′ value might be available, and then one of them is chosen randomly within particular applications.

[0249] The systematic indices for the K′ values listed in Table 2 in Section 5.6 of Appendix A is one potential list of systematic indices for the code described in Appendix A.

Variations of a Sub-blocking Process

[0250] Sub-blocking, dividing blocks into smaller units, physically or logically, for further processing, is known for various purposes. For example, it is used in IETF RFC 5053. It is also known from U.S. Patent No. 7,072,971. One of the primary uses of the sub-blocking method is to allow a large block of data to be protected as a single entity by an FEC code, while at the same time using a much smaller amount of memory than the size of the data block at a receiver to recover the data block using an FEC decoder.

[0251] One method for choosing the number of sub-blocks described in IETF RFC 5053 provides a good source block partition and sub-block partition for many reasonable settings of parameters, but it may produce a solution in some circumstances that may not strictly satisfy an upper bound on the sub-block size WS (although even in these cases it produces solutions where the sub-block size is a modest factor larger than the given constraint WS on the sub-block size). As another example, in draft-luby-rmt-bb-fec-raptorq-object-00 (where the maximum number of source symbols in a source block is much larger than in IETF RFC 5053), in Section 4.2, the recipe below is provided to calculate T, Z, and N, where T is the symbol size, Z is the number of source blocks into which the file (or data block) is partitioned, and N is the number of sub-blocks. Also, P′ is the packet payload size for symbols, F is the file size in bytes, K′_max is the maximum number of source symbols supported (e.g., 56,404), Al is an alignment factor specifying that symbols or sub-symbols should be multiples of Al bytes in size to allow more efficient decoding, e.g., Al = 4 for a modern CPU is preferred, and WS is the desired upper bound on the sub-block size in bytes.

[0252] Note that the derivation of the parameters T, Z, and N can be done at a sender or an alternative server based on the values of F, Al, and P′. The receiver only needs to know the values of F, Al, T, Z, and N in order to determine the sub-block and source block structure of the file or data block in received packets pertaining to the file or data block. The receiver can determine P′ from the size of received packets. Note that sent and received packets also typically contain other information that identifies the contents of the packet, e.g., an FEC Payload ID that is typically 4 bytes in size and that carries the source block number (SBN), and the encoded symbol identifier (ESI) of the first symbol carried in the packet.

[0253] A previous method described in Section 4.2 of draft-luby-rmt-bb-fec-raptorg-object-00 to calculate T, Z, N is to set them at the following values:

- T = P′
- Kt = ceil(F/T)
- Z = ceil(Kt/K′_max)
- N = min{ceil(ceil(Kt/Z)*T/WS), T/Al}

[0254] In these calculations, ceil() is a function that outputs the smallest integer greater than or equal to its input, and floor() is a function that outputs the largest integer less than or equal to its input. Also, min() is a function that outputs the minimum of its inputs.

[0255] One issue for some settings of parameters with this way of deriving source blocks and sub-block partitioning is that if T/Al is smaller than ceil(ceil(Kt/Z)*T/WS), then the upper bound on the sub-block size W may not be respected.

[0256] A potential secondary issue is that this allows sub-symbols to be as small as Al, which is typically set to 4 bytes, and may be too small to be efficient in practice. Typically, the smaller the sub-symbol size, the more processing overhead there is to decode or encode sub-blocks. Furthermore, especially at a receiver, a smaller sub-symbol size means that more sub-blocks need to be de-multiplexed and decoded, and this can consume receiver resources such as CPU cycles and memory accesses. On the other hand, a smaller allowable sub-symbol size means that a source block can be partitioned into more sub-blocks that respect a specified upper bound WS on sub-block size. Thus, smaller sub-symbols allow a larger source block to be supported, and thus the FEC protection provided across this source block yields better protection and better network efficiency. In practice, in many cases it is preferable to ensure that sub-symbols are at least a specified minimum size, which provides the opportunity for a better balance between processing requirements and memory requirements at a receiver and the efficient usage of network resources.

[0257] As an example of the derived parameters using the previous method described in Section 4.2 of draft-luby-rmt-bb-fec-raptorg-object-00 to calculate T, Z, N:

[0258] Input:

F = 56,404 KB
P′ = 1 KB = 1,024 bytes
WS = 128 KB
Al = 4
K′_max = 56,404

[0259] Calculations:

T = 1 KB
Kt = 56,404
Z = 1
N = 256 (due to the second input to the min function)

[0260] In this example, there will be one source block, comprising 256 sub-blocks, where each sub-block is approximately 220 KB (larger than WS) with at least some sub-blocks having sub-symbol size 4 bytes (extremely small).

[0261] A third issue is that an AL-FEC solution may not support all possible numbers of source symbols, i.e., it may only support a selected list of K′ values, where K′ is a supported number of source symbols in a source block, and then if the actual number of source symbols K desired in a source block is not among the K′ values then K is padded up to the nearest K′ value, which means that the size of the source block that is used can be somewhat larger than the calculated K value from the above.

[0262] We now describe new sub-blocking methods, which are improvements on the previous methods described above. For the purposes of description, a module for sub-blocking might take as its inputs data to be partitioned, F, and values including WS, Al, SS and P′, where the meaning of those variables is described in more detail below.

[0263] WS represents a provided constraint on the maximum size sub-block, possibly in units of bytes, that is decodable in working memory at a receiver. Al represents a memory alignment parameter. Since a receiver memory might work more efficiently if symbols and sub-symbols are aligned in memory along memory alignment boundaries, it might be useful to track Al and store values in multiples of Al bytes. For example, typically Al = 4, as many memory devices naturally address data in memory on four byte boundaries. Other values of Al are also possible, e.g., Al = 2 or Al = 8. Typically, Al might be set to the least common multiple memory alignment of all the many possible receivers. For example, if some receivers support 2-byte memory alignment but other receivers support 4-byte memory alignment, then Al = 4 would be recommended.

[0264] The parameter SS is determined based on the preferred lower bound on sub-symbol size, such that the lower bound on the sub-symbol size is SS*Al bytes. It may be preferable to have the sub-symbol size be a multiple of Al, since decoding operations are typically performed on sub-symbols.

[0265] What follows is a detailed explanation of a method for partitioning data F into Z source blocks and then the partitioning of those Z source blocks into N sub-blocks. In this description, P′ refers to a variable stored in memory (or implied) representing the available bytes within packets for symbols that are to be sent, and it is assumed that P′ is a multiple of Al. T is a variable representing the size of the symbols that are to be placed within sent packets. Other variables can be inferred from the text.

New Sub-blocking Method to Determine T, Z and N

```
• T = P′
• Kt = ceil(F/T);
• N_max = floor(T/(SS*Al));
• For all n = 1, …, N_max
    o KL(n) is the maximum K′ value supported as a possible number of source
      symbols in a source block that satisfies
        ▪ K′ ≤ WS/(Al*(ceil(T/(Al*n))));
• Z = ceil(Kt/KL(N_max));
• N = minimum n such that ceil(Kt/Z) ≤ KL(n);
```

[0266] Once these parameters have been determined, then the size of each of the Z source blocks, and the sizes of the sub-symbols of the N sub-blocks of each source block can be determined as described in IETF RFC 5053, i.e., Kt = ceil(F/T), (KL, KS, ZL, ZS) = Partition[Kt, Z], and (TL, TS, NL, NS) = Partition[T/Al, N].

[0267] Kt is the number of source symbols in the file. In a sub-block module, the Kt source symbols are partitioned into Z source blocks, ZL source blocks with KL source symbols each and ZS source blocks with KS source symbols each. Then, KL is rounded up to KL′, where KL′ is the smallest supported number of source symbols that is at least KL (and an additional KL′ - KL zero-padding symbols are added to the source block for purposes of encoding and decoding, but these additional symbols are typically not sent or received), and similarly KS is rounded up to KS′, where KS′ is the smallest supported number of source symbols that is at least KS (and an additional KS′ - KS zero-padding symbols are added to the source block for purposes of encoding and decoding, but these additional symbols are typically not sent or received).

[0268] These calculations (performed by the sub-block module, another software module, or hardware) ensure that the numbers of source symbols for the source blocks are as equal as possible, subject to the constraint that their numbers total to the number, Kt, of source symbols in the file. These calculations also ensure that the sizes of the sub-symbols for the sub-blocks are as equal as possible subject to the constraint that they are multiples of Al and that their sizes total the symbol size.

[0269] Then, the sub-symbol parameters TL, TS, NL and NS are calculated, where there are NL sub-blocks that use the larger sub-symbol size TL*Al and there are NS sub-blocks that use the smaller sub-symbol size TS*Al. The function Partition[I, J] is implemented in software or hardware and is defined as the function with an output that is a sequence of four integers (IL, IS, JL, JS), where IL = ceil(I/J), IS = floor(I/J), JL = I - IS * J, and JS = J - JL.

[0270] Some of the properties of these new methods are worth noting. A sub-block module can determine a lower bound derived on the smallest sub-symbol size used. From the above equations, it is known that TS = floor(T/(Al*N)), where TS*Al is the smallest sub-symbol size used since $TS \leq TL$. Note that the smallest sub-symbol is used when N = N_max. Using $X/(\text{floor}(Y)) \geq X/Y$ for positive X and Y, TS is at least floor(T/(Al*floor(T/(SS*Al)))), which is in turn at least floor(SS) = SS. Because of these facts, the smallest sub-symbol size produced by the partitioning method described herein will be at least TS*Al = SS*Al, as desired.

[0271] A sub-block module can determine an upper bound derived on the largest sub-block size. The largest sub-block size used is TL*Al* KL′, where KL′ is the smallest K′ value in the table above that is at least KL = ceil(Kt/Z). Note that, by the definition of N, $KL' \leq KL(N)$, and TL = ceil(T/(Al*N)). Since $KL(N) \leq WS/(Al*(\text{ceil}(T/(Al*N))))$, it follows that $WS \geq KL(N)*Al*\text{ceil}(T/(Al*N)) \geq KL'*Al*TL$.

[0272] A variable N_max can represent the largest number of sub-symbols into which a source symbol of size T can be partitioned. Setting N_max to floor(T/(SS*Al)) ensures that the smallest sub-symbol size is at least SS*Al. KL(n) is the largest number of source symbols in a source block that can be supported when symbols of the source block are partitioned into n sub-symbols each, to ensure that each of the sub-blocks of the source block is of size at most WS.

[0273] The number Z of source blocks can be chosen to be as small as possible, subject to the constraint that the number of source symbols in each source block is at most KL(N_max), which ensures that each source symbol can be partitioned into sub-symbols of size at least SS*Al and that the resulting sub-blocks are of size at most WS. The sub-block module determines, from the value of Z, the number of source blocks and the numbers of symbols in each of the Z source blocks.

[0274] Note that if any smaller value of Z is used than produced by this partitioning method, then either there would be a sub-block of one of the source blocks that is larger than WS or else there would be a sub-block of one of the source blocks that had a sub-symbol size smaller than SS*Al. Also, the smallest of the source blocks that this partitioning method produces is as large as possible subject to these two constraints, i.e., there is no other method to partition the file or data block into source blocks that respects both constraints such that the smallest source block is larger than the smallest source block produced by this partitioning method. Thus, in this sense the value of Z produced by this partitioning method is optimal.

[0275] The number N of sub-blocks into which a source block is partitioned can be chosen to be as small as possible subject to the constraint that, for each sub-block, the size of the sub-symbols of the sub-block times the number of source symbols in the source block which the sub-block partitions is at most WS.

[0276] Note that if any smaller value of N is used than produced by this partitioning method from the value of Z, then there would be at least one sub-block whose size would exceed WS. Also, the smallest sub-symbol size that this partitioning method produces from the given value of Z is as large as possible subject to the constraint that the largest sub-block size should not exceed WS, i.e., there is no other method to produce sub-blocks of the source blocks determined by the value of Z that respects the largest sub-block constraint such that the smallest sub-symbol size is larger than the smallest sub-symbol size produced by this partitioning method. Thus, in this sense the value of N produced by this partitioning method is optimal.

[0277] In the following examples, it is assumed that all possible K′ values are supported as a number of source symbols in a source block.

Example 1

[0278] Inputs:

SS = 5
Al = 4 bytes
(Min sub-symbol size = 20 bytes)
WS = 128 KB = 131,072 bytes
P′ = 1,240 bytes
F = 6 MB = 6,291,456 bytes

[0279] Calculations:

T = 1,240 bytes
Kt = 5,074
N_max = 62
KL(N_max) = 6,553
Z = 1
KL = ceil(Kt/Z) = 5,074
N = 52 (KL(N) = 5,461)
TL = 6, larger sub-symbol = 24 bytes
TS = 5, smaller sub-symbol = 20 bytes
TL * AL * KL = 121,776

Example 2

[0280] Inputs:

SS = 8
Al = 4 bytes
(Min sub-symbol size = 32 bytes)
WS = 128 KB = 131,072 bytes
P′ = 1 KB = 1,024 bytes
F = 56,404 KB = 57,757,696 bytes

[0281] Calculations:

T = 1,024 bytes
Kt = 56,404
N_max = 32
KL(N_max) = 4,096
Z = 14
KL = ceil(Kt/Z) = 4,029
N = 32 (KL(N) = 4,096)
TL = 8, larger sub-symbol = 32 bytes
TS = 8, smaller sub-symbol = 32 bytes
TL * AL * KL = 128,928

[0282] There are many variants of the above methods. For example, for some FEC code it is desirable to have at least a minimum number of source symbols in a source block to minimize the source block reception overhead of the FEC code. Since for really small file sizes or data block sizes F the size of the source symbol might become too small, there might also be a maximum number of source symbols into which a packet is partitioned. For example, in IETF RFC 5053, the desired minimal number of source symbols in a source block is Kmin = 1024 and the maximum number of source symbols into which a packet is partitioned is Gmax = 10.

[0283] Below is another variant of the new sub-blocking method described above that takes into account the additional parameters Kmin and Gmax as just described, where G is the number of symbols for a source block carried in each packet, performable by a subblocking module or more generally some module or software or hardware at an encoder, decoder, transmitter and/or receiver.

[0284] In this variant, each packet carries the ESI of the first symbol in the packet and then each subsequent symbol in the packet implicitly has an ESI that is one larger than the preceding symbol in the packet.

<u>New Sub-blocking Method to Determine G, T, Z and N</u>

- G = min(ceil(P′*Kmin/F), floor(P′/(SS*Al)), Gmax);
- T = floor(P′/(Al*G))*Al;
- Kt = ceil(F/T);
- N_max = floor(T/(SS*Al));
- For all n = 1, …, N_max
    - KL(n) is the maximum K′ value supported as a possible number of source symbols in a source block that satisfies
        - K′ ≤ WS/(Al*(ceil(T/(Al*n))));
- Z = ceil(Kt/KL(N_max));
- N = minimum n such that ceil(Kt/Z) ≤ KL(n);

[0285] Note that by the way G is calculated, it is guaranteed that the symbol size is at least SS*Al, i.e., the symbol size is at least the minimum sub-symbol size. Note also that it should be the case that SS*Al is at least P′ to ensure that the symbol size can be at least SS*Al (and if it is not, then G will evaluate to zero).

<u>Example 3</u>

[0286] Input:

SS = 5
Al = 4 bytes
(Min sub-symbol size = 20 bytes)
WS = 256 KB = 262,144 bytes

P′ = 1,240 bytes
F = 500 KB = 512,000 bytes
Kmin = 1,024
Gmax = 10

[0287] Calculations:

G = 3
T = 412
Kt = 1,243
N_max = 20
KL(N_max) = 10,992
Z = 1
KL = ceil(Kt/Z) = 1,243
N = 2 (KL(N) = 1,260)
TL = 52, larger sub-symbol = 208 bytes
TS = 51, smaller sub-symbol = 204 bytes
TL * AL * KL = 258,544

[0288] As has now been described, these new methods introduce a constraint on the smallest sub-symbol size used for any sub-block. This disclosure provides new methods for sub-blocking that respect this additional constraint, while at the same time strictly respecting a constraint on the maximum sub-block size. The methods produce source blocking and sub-blocking solutions that satisfy the objectives of partitioning a file or data block into as few source blocks as possible subject to a constraint on the smallest sub-symbol size, and then subject to this split into as few sub-blocks as possible subject to a constraint on the maximum sub-block size.

Variations

[0289] In some applications, it may be acceptable to not be able to decode all of the source symbols, or to be able to decode all of source symbols, but with a relatively low probability. In such applications, a receiver can stop attempting to decode all of the source symbols after receiving K+A encoded symbols. Or, the receiver can stop receiving encoded symbols after receiving less than K+A encoded symbols. In some applications, the receiver may even only receive K or less encoded symbols. Thus, it is to be understood that in some embodiments of the present invention, the desired degree of accuracy need not be complete recovery of all the source symbols.

[0290] Further, in some applications where incomplete recovery is acceptable, the data can be encoded such that all of the source symbols cannot be recovered, or such that complete recovery of the source symbols would require reception of many more encoded symbols than the number of source symbols. Such an encoding would generally require less computational expense, and may thus be an acceptable way to decrease the computational expense of encoding.

[0291] It is to be understood that the various functional blocks in the above-described figures may be implemented by a combination of hardware and/or software, and that in specific implementations some or all of the functionality of some of the blocks may be combined. Similarly, it is also to be understood that the various methods taught herein may be implemented by a combination of hardware and/or software.

[0292] The above description is illustrative and not restrictive. Many variations of the invention will become apparent to those of skill in the art upon review of this disclosure. The scope of the invention should, therefore, be determined not with reference to the above description, but instead should be determined with reference to the appended claims along with their full scope of equivalents.

APPENDIX A.

```
Reliable Multicast Transport                                    M. Luby
Internet-Draft                                    Qualcomm Incorporated
Intended status: Standards Track           .             A. Shokrollahi
Expires: February 12, 2011                  .                      EPFL
                                             .                M. Watson
                                              .   Qualcomm Incorporated
                                               .         T. Stockhammer
                                                .        Nomor Research
                                                 .            L. Minder
                                                  .Qualcomm Incorporated
                                                   ..   August 11, 2010
```

RaptorQ Forward Error Correction Scheme for Object Delivery
draft-ietf-rmt-bb-fec-raptorq-04

Abstract

This document describes a Fully-Specified FEC scheme, corresponding to FEC Encoding ID 6 (to be confirmed (tbc)), for the RaptorQ forward error correction code and its application to reliable delivery of data objects.

RaptorQ codes are a new family of codes that provide superior flexibility, support for larger source block sizes and better coding efficiency than Raptor codes in RFC5053. RaptorQ is also a fountain code, i.e., as many encoding symbols as needed can be generated by the encoder on-the-fly from the source symbols of a source block of data. The decoder is able to recover the source block from any set of encoding symbols for most cases equal to the number of source symbols and in rare cases with slightly more than the number of source symbols.

The RaptorQ code described here is a systematic code, meaning that all the source symbols are among the encoding symbols that can be generated.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

This document specifies an FEC Scheme for the RaptorQ forward error correction code for object delivery applications. The concept of an FEC Scheme is defined in RFC5052 [RFC5052] and this document follows the format prescribed there and uses the terminology of that document. The RaptorQ code described herein is a next generation of the Raptor code described in RFC5053 [RFC5053]. The RaptorQ code provides superior reliability, better coding efficiency, and support for larger source block sizes than the Raptor code of RFC5053 [RFC5053]. These improvements simplify the usage of the RaptorQ code in an object delivery Content Delivery Protocol compared to RFC5053 [RFC5053].

The RaptorQ FEC Scheme is a Fully-Specified FEC Scheme corresponding to FEC Encoding ID 6 (tbc).

> Editor's Note: The finalized FEC encoding ID is still to be defined, but '6 (tbc)' is used as temporary value in this Internet Draft expecting sequential use of FEC encoding IDs in the IANA registration process.

RaptorQ is a fountain code, i.e., as many encoding symbols as needed can be generated by the encoder on-the-fly from the source symbols of a block. The decoder is able to recover the source block from any set of encoding symbols only slightly more in number than the number of source symbols.

The code described in this document is a systematic code, that is, the original source symbols can be sent unmodified from sender to receiver, as well as a number of repair symbols. For more backgound on the use of Forward Error Correction codes in reliable multicast, see [RFC3453].

2. Requirements notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Formats and Codes

3.1. FEC Payload IDs

The FEC Payload ID MUST be a 4-octet field defined as follows:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     SBN       |               Encoding Symbol ID              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1: FEC Payload ID format

o Source Block Number (SBN), (8 bits, unsigned integer): A non-negative integer identifier for the source block that the encoding symbols within the packet relate to.

o Encoding Symbol ID (ESI), (24 bits, unsigned integer): A non-negative integer identifier for the encoding symbols within the packet.

The interpretation of the Source Block Number and Encoding Symbol Identifier is defined in Section 4.

3.2. FEC Object Transmission Information

3.2.1. Mandatory

The value of the FEC Encoding ID MUST be 6, as assigned by IANA (see Section 7).

3.2.2. Common

The Common FEC Object Transmission Information elements used by this FEC Scheme are:

o Transfer Length (F), (40 bits, unsigned integer): A non-negative integer that is at most 946270874880. This is the transfer length of the object in units of octets.

o Symbol Size (T), (16 bits, unsigned integer): A positive integer that is less than 2^^16. This is the size of a symbol in units of octets.

The encoded Common FEC Object Transmission Information format is shown in Figure 2.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Transfer Length (F)                      |
+               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               |    Reserved     |        Symbol Size (T)      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2: Encoded Common FEC OTI for RaptorQ FEC Scheme

NOTE 1: The limit of 946270874880 on the transfer length is a consequence of the limitation on the symbol size to 2^^16-1, the limitation on the number of symbols in a source block to 56403 and the limitation on the number of source blocks to 2^^8.

3.2.3. Scheme-Specific

The following parameters are carried in the Scheme-Specific FEC Object Transmission Information element for this FEC Scheme:

o The number of source blocks (Z) (12 bits, unsigned integer)

o The number of sub-blocks (N) (12 bits, unsigned integer)

o A symbol alignment parameter (Al) (8 bits, unsigned integer)

These parameters are all positive integers. The encoded Scheme-specific Object Transmission Information is a 4-octet field consisting of the parameters Z, N and Al as shown in Figure 3.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Z         |              N        |       Al      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3: Encoded Scheme-specific FEC Object Transmission Information

The encoded FEC Object Transmission Information is a 12-octet field consisting of the concatenation of the encoded Common FEC Object Transmission Information and the encoded Scheme-specific FEC Object Transmission Information.

These three parameters define the source block partitioning as described in Section 4.4.1.2

4. Procedures

4.1. Introduction

For any undefined symbols or functions used in this section, in particular the functions "ceil" and "floor", refer to Section 5.1.

4.2. Content Delivery Protocol Requirements

This section describes the information exchange between the RaptorQ FEC Scheme and any Content Delivery Protocol (CDP) that makes use of the RaptorQ FEC Scheme for object delivery.

The RaptorQ encoder scheme and RaptorQ decoder scheme for object delivery require the following information from the CDP:

o The transfer length of the object, F, in octets

o A symbol alignment parameter, Al

o The symbol size, T, in octets, which MUST be a multiple of Al

o The number of source blocks, Z

o The number of sub-blocks in each source block, N

The RaptorQ encoder scheme for object delivery additionally requires:

- the object to be encoded, F octets

The RaptorQ encoder scheme supplies the CDP with the following information for each packet to be sent:

o Source Block Number (SBN)

o Encoding Symbol ID (ESI)

o Encoding symbol(s)

The CDP MUST communicate this information to the receiver.

4.3. Example Parameter Derivation Algorithm

This section provides recommendations for the derivation of the three transport parameters, T, Z and N. This recommendation is based on the following input parameters:

o F the transfer length of the object, in octets

o WS the maximum size block that is decodable in working memory, in octets

o P' the maximum payload size in octets, which is assumed to be a multiple of Al

o Al the symbol alignment parameter, in octets

o SS a parameter where the desired lower bound on the sub-symbol size is SS*Al

o K'_max the maximum number of source symbols per source block.

Note: Section 5.1.2 defines K'_max to be 56403.

Based on the above inputs, the transport parameters T, Z and N are calculated as follows:

Let,

o T = P'

o Kt = ceil(F/T)

o N_max = floor(T/(SS*Al))

o for all n=1, ..., N_max

  * KL(n) is the maximum K' value in Table 2 in Section 5.6 such that

    K' <= WS/(Al*(ceil(T/(Al*n))))

o Z = ceil(Kt/KL(N_max))

o N is the minimum n=1, ..., N_max such that ceil(Kt/Z) <= KL(n)

It is RECOMMENDED that each packet contains exactly one symbol. However, receivers SHALL support the reception of packets that contain multiple symbols.

The value Kt is the total number of symbols required to represent the source data of the object.

The algorithm above and that defined in Section 4.4.1.2 ensure that the sub-symbol sizes are a multiple of the symbol alignment parameter, Al. This is useful because the sum operations used for encoding and decoding are generally performed several octets at a time, for example at least 4 octets at a time on a 32 bit processor. Thus the encoding and decoding can be performed faster if the sub-symbol sizes are a multiple of this number of octets.

The recommended setting for the input parameter Al is 4.

The parameter WS can be used to generate encoded data which can be decoded efficiently with limited working memory at the decoder. Note that the actual maximum decoder memory requirement for a given value of WS depends on the implementation, but it is possible to implement decoding using working memory only slightly larger than WS.

4.4. Object Delivery

4.4.1. Source block construction

4.4.1.1. General

In order to apply the RaptorQ encoder to a source object, the object may be broken into Z >= 1 blocks, known as source blocks. The RaptorQ encoder is applied independently to each source block. Each source block is identified by a unique Source Block Number (SBN), where the first source block has SBN zero, the second has SBN one, etc. Each source block is divided into a number, K, of source symbols of size T octets each. Each source symbol is identified by a unique Encoding Symbol Identifier (ESI), where the first source symbol of a source block has ESI zero, the second has ESI one, etc.

Each source block with K source symbols is divided into N >= 1 sub-blocks, which are small enough to be decoded in the working memory. Each sub-block is divided into K sub-symbols of size T'.

Note that the value of K is not necessarily the same for each source block of an object and the value of T' may not necessarily be the same for each sub-block of a source block. However, the symbol size T is the same for all source blocks of an object and the number of symbols, K is the same for every sub-block of a source block. Exact partitioning of the object into source blocks and sub-blocks is described in Section 4.4.1.2 below.

4.4.1.2. Source block and sub-block partitioning

The construction of source blocks and sub-blocks is determined based on five input parameters, F, Al, T, Z and N and a function Partition[]. The five input parameters are defined as follows:

- o F the transfer length of the object, in octets
- o Al a symbol alignment parameter, in octets
- o T the symbol size, in octets, which MUST be a multiple of Al
- o Z the number of source blocks
- o N the number of sub-blocks in each source block

These parameters MUST be set so that ceil(ceil(F/T)/Z) <= K'_max. Recommendations for derivation of these parameters are provided in Section 4.3.

The function Partition[] takes a pair of positive integers (I, J) as input and derives four non-negative integers (IL, IS, JL, JS) as output. Specifically, the value of Partition[I, J] is the sequence (IL, IS, JL, JS), where IL = ceil(I/J), IS = floor(I/J), JL = I - IS * J and JS = J - JL. Partition[] derives parameters for partitioning a block of size I into J approximately equal sized blocks. Specifically, JL blocks of length IL and JS blocks of length IS.

The source object MUST be partitioned into source blocks and sub-blocks as follows:

Let

- o Kt = ceil(F/T),
- o (KL, KS, ZL, ZS) = Partition[Kt, Z],
- o (TL, TS, NL, NS) = Partition[T/Al, N].

Then, the object MUST be partitioned into Z = ZL + ZS contiguous source blocks, the first ZL source blocks each having KL*T octets, i.e. KL source symbols of T octets each, and the remaining ZS source blocks each having KS*T octets, i.e. KS source symbols of T octets each.

If Kt*T > F then for encoding purposes, the last symbol of the last source block MUST be padded at the end with Kt*T-F zero octets.

Next, each source block with K source symbols MUST be divided into N = NL + NS contiguous sub-blocks, the first NL sub-blocks each consisting of K contiguous sub-symbols of size of TL*Al octets and the remaining NS sub-blocks each consisting of K contiguous sub-symbols of size of TS*Al octets. The symbol alignment parameter Al ensures that sub-symbols are always a multiple of Al octets.

Finally, the m-th symbol of a source block consists of the concatenation of the m-th sub-symbol from each of the N sub-blocks. Note that this implies that when N > 1 then a symbol is NOT a contiguous portion of the object.

4.4.2. Encoding packet construction

Each encoding packet contains the following information:

- o Source Block Number (SBN)
- o Encoding Symbol ID (ESI)
- o encoding symbol(s)

Each source block is encoded independently of the others. Source blocks are numbered consecutively from zero.

Encoding Symbol ID values from 0 to K-1 identify the source symbols of a source block in sequential order, where K is the number of source symbols in the source block. Encoding Symbol IDs K onwards identify repair symbols generated from the source symbols using the RaptorQ encoder.

Each encoding packet either consists entirely of source symbols (source packet) or entirely of repair symbols (repair packet). A packet may contain any number of symbols from the same source block. In the case that the last source symbol in a source packet includes padding octets added for FEC encoding purposes then these octet need not be included in the packet. Otherwise, only whole symbols MUST be included.

The Encoding Symbol ID, X, carried in each source packet is the Encoding Symbol ID of the first source symbol carried in that packet. The subsequent source symbols in the packet have Encoding Symbol IDs, X+1 to X+G-1, in sequential order, where G is the number of symbols in the packet.

Similarly, the Encoding Symbol ID, X, placed into a repair packet is the Encoding Symbol ID of the first repair symbol in the repair packet and the subsequent repair symbols in the packet have Encoding Symbol IDs X+1 to X+G-1 in sequential order, where G is the number of symbols in the packet.

Note that it is not necessary for the receiver to know the total number of repair packets.

5. RaptorQ FEC Code Specification

5.1. Definitions, Symbols and Abbreviations

For the purpose of the RaptorQ FEC code specification in this section, the following definitions, symbols and abbreviations apply.

5.1.1. Definitions

o Source block: a block of K source symbols which are considered together for RaptorQ encoding and decoding purposes.

o Extended Source Block: a block of K' source symbols, where K' >= K constructed from a source block and zero or more padding symbols.

o Symbol: a unit of data. The size, in octets, of a symbol is known as the symbol size. The symbol size is always a positive integer.

o Source symbol: the smallest unit of data used during the encoding process. All source symbols within a source block have the same size.

o Padding symbol: a symbol with all zero bits that is added to the source block to form the extended source block.

o Encoding symbol: a symbol that can be sent as part of the encoding of a source block. The encoding symbols of a source block consist of the source symbols of the source block and the repair symbols generated from the source block. Repair symbols generated from a source block have the same size as the source symbols of that source block.

o Repair symbol: the encoding symbols of a source block that are not source symbols. The repair symbols are generated based on the source symbols of a source block.

o Intermediate symbols: symbols generated from the source symbols using an inverse encoding process. The repair symbols are then generated directly from the intermediate symbols. The encoding symbols do not include the intermediate symbols, i.e., intermediate symbols are not sent as part of the encoding of a source block. The intermediate symbols are partitioned into LT symbols and PI symbols.

o LT symbols: The subset of the intermediate symbols that can be LT neighbors of an encoding symbol.

- PI symbols: The subset of the intermediate symbols that can be PI neighbors of an encoding symbol.

- Systematic code: a code in which all source symbols are included as part of the encoding symbols of a source block. The RaptorQ code as described herein is a systematic code.

- Encoding Symbol ID (ESI): information that uniquely identifies each encoding symbol associated with a source block for sending and receiving purposes.

- Internal Symbol ID (ISI): information that uniquely identifies each symbol associated with an extended source block for encoding and decoding purposes.

- Arithmetic operations on octets and symbols and matrices: The operations that are used to produce encoding symbols from source symbols and vice-versa. See Section 5.7.

5.1.2. Symbols

i, j, u, v, h, d, a, b, d1, a1, b1, v, m, x, y represent values or variables of one type or another, depending on the context.

X denotes a non-negative integer value that is either an ISI value or an ESI value, depending on the context.

ceil(x) denotes the smallest integer which is greater than or equal to x, where x is a real value.

floor(x) denotes the largest integer which is less than or equal to x, where x is a real value.

min(x,y) denotes the minimum value of the values x and y, and in general the minimum value of all the argument values.

max(x,y) denotes the maximum value of the values x and y, and in general the maximum value of all the argument values.

i % j denotes i modulo j.

i + j denotes the sum of i and j. If i and j are octets, respectively symbols, this designates the arithmetic on octets, respectively symbols, as defined in Section 5.7. If i and j are integers, then it denotes the usual integer addition.

i * j denotes the product of i and j. If i and j are octets, this designates the arithmetic on octets, as defined in Section 5.7. If i is an octet and j is a symbol, this denotes the multiplication of a symbol by an octet, as also defined in Section 5.7. Finally, if i and j are integers, i * j denotes the usual product of integers.

a ^^ b denotes the operation a raised to the power b. If a is an octet and b is a non-negative integer, this is understood to mean a*a*...*a (b terms), with '*' being the octet product as defined in Section 5.7.

u ^ v denotes, for equal-length bit strings u and v, the bitwise exclusive-or of u and v.

Transpose[A] denotes the transposed matrix of matrix A. In this specification, all matrices have entries that are octets.

A^^-1 denotes the inverse matrix of matrix A. In this specification, all the matrices have octets as entries, so it is understood that the operations of the matrix entries are to be done as stated in Section 5.7 and A^^-1 is the matrix inverse of A with respect to octet arithmetic.

K denotes the number of symbols in a single source block.

K' denotes the number of source plus padding symbols in an extended source block. For the majority of this specification, the padding symbols are considered to be additional source symbols.

K'_max denotes the maximum number of source symbols that can be in a single source block. Set to 56403.

L denotes the number of intermediate symbols for a single extended source block.

S denotes the number of LDPC symbols for a single extended source block. These are LT symbols. For each value of K' shown in Table 2 in Section 5.6, the corresponding value of S is a prime number.

H denotes the number of HDPC symbols for a single extended source block. These are PI symbols.

B denotes the number of intermediate symbols that are LT symbols excluding the LDPC symbols.

W denotes the number of intermediate symbols that are LT symbols. For each value of K' in Table 2 shown in Section 5.6, the corresponding value of W is a prime number.

P denotes the number of intermediate symbols that are PI symbols. These contain all HDPC symbols.

P1 denotes the smallest prime number greater than or equal to P.

U denotes the number of non-HDPC intermediate symbols that are PI symbols.

C denotes an array of intermediate symbols, C[0], C[1], C[2],..., C[L-1].

C' denotes an array of the symbols of the extended source block, where C'[0], C'[1], C'[2],..., C'[K-1] are the source symbols of the source block and C'[K], C'[K+1],..., C'[K'-1] are padding symbols.

V0, V1, V2, V3 denote four arrays of 32-bit unsigned integers, V0[0], V0[1],..., V0[255] ; V1[0], V1[1],..., V1[255]; V2[0], V2[1],..., V2[255]; and V3[0], V3[1],..., V3[255] as shown in Section 5.5.

Rand[y, i, m] denotes a pseudo-random number generator

Deg[v] denotes a degree generator

Enc[K', C ,(d, a, b, d1, a1, b1)] denotes an encoding symbol generator

Tuple[K', X] denotes a tuple generator function

T denotes the symbol size in octets.

J(K') denotes the systematic index associated with K'.

G denotes any generator matrix.

I_S denotes the SxS identity matrix.

5.1.3. Abbreviations

ESI	Encoding Symbol ID
HDPC	High Density Parity Check
ISI	Internal Symbol ID
LDPC	Low Density Parity Check
LT	Luby Transform
PI	Permanently Inactive
SBN	Source Block Number
SBL	Source Block Length (in units of symbols)

5.2. Overview

This section defines the systematic RaptorQ FEC code.

Symbols are the fundamental data units of the encoding and decoding process. For each source block all symbols are the same size, referred to as the symbol size T. The atomic operations performed on symbols for both encoding and decoding are the arithmetic operations defined in Section 5.7.

The basic encoder is described in Section 5.3. The encoder first derives a block of intermediate symbols from the source symbols of a source block. This intermediate block has the property that both source and repair symbols can be generated from it using the same process. The encoder produces repair symbols from the intermediate block using an efficient process, where each such repair symbol is the exclusive OR of a small number of intermediate symbols from the block. Source symbols can also be reproduced from the intermediate block using the same process. The encoding symbols are the combination of the source and repair symbols.

An example of a decoder is described in Section 5.4. The process for producing source and repair symbols from the intermediate block is designed so that the intermediate block can be recovered from any sufficiently large set of encoding symbols, independent of the mix of source and repair symbols in the set. Once the intermediate block is recovered, missing source symbols of the source block can be recovered using the encoding process.

Requirements for a RaptorQ compliant decoder are provided in Section 5.8. A number of decoding algorithms are possible to achieve these requirements. An efficient decoding algorithm to achieve these requirements is provided in Section 5.4.

The construction of the intermediate and repair symbols is based in part on a pseudo-random number generator described in Section 5.3. This generator is based on a fixed set of 1024 random numbers which must be available to both sender and receiver. These numbers are provided in Section 5.5. Encoding and decoding operations for RaptorQ use operations on octets. Section 5.7 describes how to perform these operations.

Finally, the construction of the intermediate symbols from the source symbols is governed by "systematic indices", values of which are provided in Section 5.6 for specific extended source block sizes between 6 and K'_max = 56403 source symbols. Thus, the RaptorQ code supports source blocks with between 1 and 56403 source symbols.

5.3. Systematic RaptorQ encoder

5.3.1. Introduction

For a given source block of K source symbols, for encoding and decoding purposes the source block is augmented with K'-K additional padding symbols, where K' is the smallest value that is at least K in the systematic index Table 2 of Section 5.6. The reason for padding out a source block to a multiple of K' is to enable faster encoding and decoding, and to minimize the amount of table information that needs to be stored in the encoder and decoder.

For purposes of transmitting and receiving data, the value of K is used to determine the number of source symbols in a source block, and thus K needs to be known at the sender and the receiver. In this case the sender and receiver can compute K' from K and the K'-K padding symbols can be automatically added to the source block without any additional communication. The encoding symbol ID (ESI) is used by a sender and receiver to identify the encoding symbols of a source block, where the encoding symbols of a source block consist of the source symbols and the repair symbols associated with the source block. For a source block with K source symbols, the ESIs for the source symbols are 0,1,2,...,K-1 and the ESIs for the repair symbols are K, K+1, K+2,... . Using the ESI for identifying encoding symbols in transport ensures that the ESI values continue consecutively between the source and repair symbols.

For purposes of encoding and decoding data, the value of K' derived from K is used as the number of source symbols of the extended source block upon which encoding and decoding operations are performed, where the K' source symbols consist of the original K source symbols and an additional K'-K padding symbols. The internal symbol ID (ISI) is used by the encoder and decoder to identify the symbols associated with the extended source block, i.e., for generating encoding symbols and for decoding. For a source block with K original source symbols, the ISIs for the original source symbols are 0,1,2,...,K-1, the ISIs for the K'-K padding symbols are K, K+1, K+2,..., K'-1, and the ISIs for the repair symbols are K', K'+1, K'+2,... . Using the ISI for encoding and decoding allows the padding symbols of the extended source block to be treated the same way as other source symbols of the extended source block, and that a given prefix of repair symbols are generated in a consistent way for a given number K' of source symbols in the extended source block independent of K.

The relationship between the ESIs and the ISIs is simple: the ESIs and the ISIs for the original K source symbols are the same, the K'-K padding symbols have an ISI but do not have a corresponding ESI (since they are symbols that are neither sent nor received), and a repair symbol ISI is simply the repair symbol ESI plus K'-K. The translation between ESIs used to identify encoding symbols sent and received and the corresponding ISIs used for encoding and decoding, and the proper padding of the extended source block with padding symbols used for encoding and decoding, is the responsibility of the padding function in the RaptorQ encoder/decoder.

5.3.2. Encoding overview

The systematic RaptorQ encoder is used to generate any number of repair symbols from a source block that consists of K source symbols placed into an extended source block C'. Figure 4 shows the encoding overview.

The first step of encoding is to construct an extended source block by adding zero or more padding symbols such that the total number of symbols, K', is one of the values listed in Section 5.6. Each padding symbol consists of T octets where the value of each octet is zero. K' MUST be selected as the smallest value of K' from the table of Section 5.6 which is greater than or equal to K.

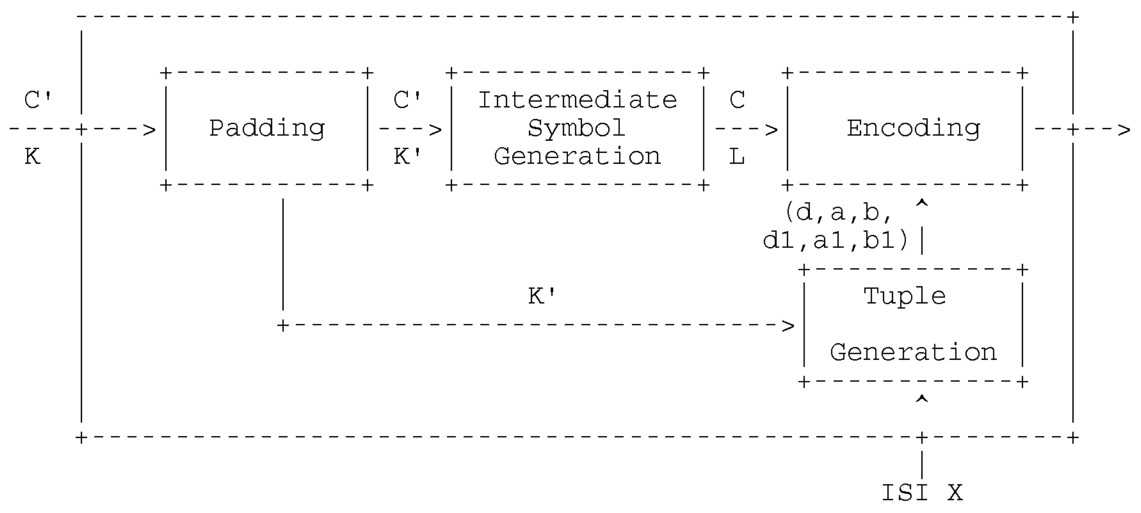


Figure 4: Encoding Overview

Let C'[0], ..., C'[K-1] denote the K source symbols.

Let C'[K], ..., C'[K'-1] denote the K'-K padding symbols, which are all set to zero bits. Then, C'[0],..., C'[K'-1] are the symbols of the extended source block upon which encoding and decoding are performed.

In the remainder of this description these padding symbols will be considered as additional source symbols and referred to as such. However, these padding symbols are not part of the encoding symbols, i.e., they are not sent as part of the encoding. At a receiver, the value of K' can be computed based on K, then the receiver can insert K'-K padding symbols at the end of a source block of K' source symbols and recover the remaining K source symbols of the source block from received encoding symbols.

The second step of encoding is to generate a number, L > K', of intermediate symbols from the K' source symbols. In this step, K' source tuples (d[0], a[0], b[0], d1[0], a1[0], b1[0]), ..., (d[K'-1], a[K'-1], b[K'-1], d1[K'-1], a1[K'-1], b1[K'-1]) are generated using the Tuple[] generator as described in Section 5.3.5.4. The K' source tuples and the ISIs associated with the K' source symbols are used to determine L intermediate symbols C[0],..., C[L-1] from the source symbols using an inverse encoding process. This process can be realized by a RaptorQ decoding process.

Certain "pre-coding relationships" must hold within the L intermediate symbols. Section 5.3.3.3 describes these relationships. Section 5.3.3.4 describes how the intermediate symbols are generated from the source symbols.

Once the intermediate symbols have been generated, repair symbols can be produced. For a repair symbol with ISI X>K', the tuple of non-negative integers, (d, a, b, d1, a1, b1) can be generated, using the Tuple[] generator as described in Section 5.3.5.4. Then, the (d, a, b, d1, a1, b1)-tuple and the ISI X is used to generate the corresponding repair symbol from the intermediate symbols using the Enc[] generator described in Section 5.3.5.3. The corresponding ESI for this repair symbol is then X-(K'-K). Note that source symbols of the extended source block can also be generated using the same process, i.e., for any X < K', the symbol generated using this process has the same value as C'[X].

5.3.3. First encoding step: Intermediate Symbol Generation

5.3.3.1. General

This encoding step is a pre-coding step to generate the L intermediate symbols C[0], ..., C[L-1] from the source symbols C'[0], ..., C'[K'-1], , where L > K' is defined in Section 5.3.3.3. The intermediate symbols are uniquely defined by two sets of constraints:

1. The intermediate symbols are related to the source symbols by a set of source symbol tuples and by the ISIs of the source symbols. The generation of the source symbol tuples is defined in Section 5.3.3.2 using the the Tuple[] generator as described in Section 5.3.5.4.

2. A number of pre-coding relationships hold within the intermediate symbols themselves. These are defined in Section 5.3.3.3

The generation of the L intermediate symbols is then defined in Section 5.3.3.4.

5.3.3.2. Source symbol tuples

Each of the K' source symbols is associated with a source symbol tuple (d[X], a[X], b[X], d1[X], a1[X], b1[X]) for 0 <= X < K'. The source symbol tuples are determined using the Tuple generator defined in Section 5.3.5.4 as:

```
   For each X, 0 <= X < K'
```

(d[X], a[X], b[X], d1[X], a1[X], b1[X]) = Tuple[K, X]

5.3.3.3. Pre-coding relationships

The pre-coding relationships amongst the L intermediate symbols are defined by requiring that a set of S+H linear combinations of the intermediate symbols evaluate to zero. There are S LDPC and H HDPC symbols, and thus L = K'+S+H. Another partition of the L intermediate symbols is into two sets, one set of W LT symbols and another set of P PI symbols, and thus it is also the case that L = W+P. The P PI symbols are treated differently than the W LT symbols in the encoding process. The P PI symbols consist of the H HDPC symbols together with a set of U = P-H of the other K' intermediate symbols. The W LT symbols consist of the S LDPC symbols together with W-S of the other K' intermediate symbols. The values of these parameters are determined from K' as described below where H(K'), S(K'), and W(K') are derived from Table 2 in Section 5.6.

Let

o S = S(K')

o H = H(K')

o W = W(K')

o L = K' + S + H

o P = L - W

o P1 denote the smallest prime number greater than or equal to P

o U = P - H

o B = W - S

o C[0], ..., C[B-1] denote the intermediate symbols that are LT symbols but not LDPC symbols.

o C[B], ..., C[B+S-1] denote the S LDPC symbols that are also LT symbols.

o C[W], ..., C[W+U-1] denote the intermediate symbols that are PI symbols but not HDPC symbols.

o C[L-H], ..., C[L-1] denote the H HDPC symbols that are also PI symbols.

The first set of pre-coding relations, called LDPC relations, is described below and requires that at the end of this process the set of symbols D[0] , ..., D[S-1] are all zero:

```
o  Initialize the symbols D[0] = C[B], ..., D[S-1] = C[B+S-1].

o  For i = 0, ..., B-1 do

   *  a = 1 + floor(i/S)

   *  b = i % S

   *  D[b] = D[b] + C[i]

   *  b = (b + a) % S

   *  D[b] = D[b] + C[i]

   *  b = (b + a) % S

   *  D[b] = D[b] + C[i]

o  For i = 0, ..., S-1 do

   *  a = i % P

   *  b = (i+1) % P

   *  D[i] = D[i] + C[W+a] + C[W+b]
```

Recall that the addition of symbols is to be carried out as specified in Section 5.7.

Note that the LDPC relations as defined in the algorithm above are linear, so there exists an S x B matrix G_LDPC,1 and an S x P matrix G_LDPC,2 such that

```
   G_LDPC,1 * Transpose[(C[0],...., C[B-1])] + G_LDPC,2 *
   Transpose(C[W], ..., C[W+P-1]) + Transpose[(C[B], ..., C[B+S-1])]
   = 0
```

(The matrix G_LDPC,1 is defined by the first loop in the above algorithm, and G_LDPC,2 can be deduced from the second loop.)

The second set of relations among the intermediate symbols C[0], ..., C[L-1] are the HDPC relations and they are defined as follows:

Let

o alpha denote the octet represented by integer 2 as defined in Section 5.7.

o MT denote an H x (K' + S) matrix of octets, where for j=0,...,K'+S-2 the entry MT[i,j] is the octet represented by the integer 1 if i= Rand[j+1,6,H] or i = (Rand[j+1,6,H] + Rand[j+ 1,7,H-1] + 1) % H and MT[i,j] is the zero element for all other values of i, and for j=K'+S-1, MT[i,j] = alpha^^i for i=0,...,H-1.

o GAMMA denote a (K'+S) x (K'+S) matrix of octets, where

```
GAMMA[i,j] =

   alpha ^^ (i-j) for i >= j,

   0 otherwise.
```

Then the relationship between the first K'+S intermediate symbols C[0], ..., C[K'+S-1] and the H HDPC symbols C[K'+S], ..., C[K'+S+H-1] is given by:

```
Transpose[C[K'+S], ..., C[K'+S+H-1]] + MT * GAMMA *
Transpose[C[0], ..., C[K'+S-1]] = 0,
```

where '*' represents standard matrix multiplication utilizing the octet multiplication to define the multiplication between a matrix of octets and a matrix of symbols (in particular the column vector of symbols) and '+' denotes addition over octet vectors.

5.3.3.4. Intermediate symbols

5.3.3.4.1. Definition

Given the K' source symbols C'[0], C'[1],..., C'[K'-1] the L intermediate symbols C[0], C[1],..., C[L-1] are the uniquely defined symbol values that satisfy the following conditions:

1. The K' source symbols C'[0], C'[1],..., C'[K'-1] satisfy the K' constraints

   ```
   C'[X] = Enc[K', (C[0],..., C[L-1]), (d[X], a[X], b[X], d1[X],
   a1[X], b1[X])], for all X, 0 <= X < K',
   ```

   where (d[X], a[X], b[X], d1[X], a1[X], b1[X])) = Tuple[K',X], Tuple[] is defined in Section 5.3.5.4 and Enc[] is described in Section 5.3.5.3.

2. The L intermediate symbols C[0], C[1],..., C[L-1] satisfy the pre-coding relationships defined in Section 5.3.3.3

5.3.3.4.2. Example method for calculation of intermediate symbols

This section describes a possible method for calculation of the L intermediate symbols C[0], C[1],..., C[L-1] satisfying the constraints in Section 5.3.3.4.1

The L intermediate symbols can be calculated as follows:

Let

o C denote the column vector of the L intermediate symbols, C[0], C[1],..., C[L-1].

o D denote the column vector consisting of S+H zero symbols followed by the K' source symbols C'[0], C'[1], ..., C'[K'-1].

Then the above constraints define an L x L matrix A of octets such that:

```
A*C = D
```

The matrix A can be constructed as follows:

Let:

o G_LDPC,1 and G_LDPC,2 be S x B and S x P matrices as defined in Section 5.3.3.3.

o G_HDPC be the H x (K'+S) matrix such that

```
G_HDPC * Transpose(C[0], ..., C[K'+S-1]) = Transpose(C[K'+S],
..., C[L-1]),

i.e.  G_HDPC = MT*GAMMA
```

o I_S be the S x S identity matrix

o I_H be the H x H identity matrix

o G_ENC be the K' x L matrix such that

```
G_ENC * Transpose[(C[0], ..., C[L-1])] =
Transpose[(C'[0],C'[1],...,C'[K'-1])],
```

i.e. G_ENC[i,j] = 1 if and only if C[j] is included in the symbols which are summed to produce Enc[K', (C[0], ..., C[L-1]), (d[i], a[i], b[i], d1[i], a1[i], b1[i])] and G_ENC[i,j] = 0 otherwise.

Then:

- The first S rows of A are equal to G_LDPC,1 | I_S | G_LDPC,2.
- The next H rows of A are equal to G_HDPC | I_H.
- The remaining K' rows of A are equal to G_ENC.

The matrix A is depicted in Figure (Figure 5) below:

```
                 B               S         U         H
      +-----------------------+-------+------------------+
      |                       |       |                  |
    S |        G_LDPC,1       |  I_S  |      G_LDPC,2    |
      |                       |       |                  |
      +-----------------------+-------+----------+-------+
      |                                          |       |
    H |                G_HDPC                    |  I_H  |
      |                                          |       |
      +------------------------------------------+-------+
      |                                                  |
      |                                                  |
   K' |                      G_ENC                       |
      |                                                  |
      |                                                  |
      +--------------------------------------------------+
```

Figure 5: The matrix A

The intermediate symbols can then be calculated as:

C = (A^^-1)*D

The source tuples are generated such that for any K' matrix A has full rank and is therefore invertible. This calculation can be realized by applying a RaptorQ decoding process to the K' source symbols C'[0], C'[1],..., C'[K'-1] to produce the L intermediate symbols C[0], C[1],..., C[L-1].

To efficiently generate the intermediate symbols from the source symbols, it is recommended that an efficient decoder implementation such as that described in Section 5.4 be used.

5.3.4. Second encoding step: Encoding

In the second encoding step, the repair symbol with ISI X (X >= K') is generated by applying the generator Enc[K', (C[0], C[1],..., C[L-1]), (d, a, b, d1, a1, b1)] defined in Section 5.3.5.3 to the L intermediate symbols C[0], C[1],..., C[L-1] using the tuple (d, a, b, d1, a1, b1)=Tuple[K',X].

5.3.5. Generators

5.3.5.1. Random Number Generator

The random number generator Rand[y, i, m] is defined as follows, where y is a non-negative integer, i is a non-negative integer less than 256, and m is a positive integer and the value produced is an integer between 0 and m-1. Let V0, V1, V2 and V3 be the arrays provided in Section 5.5.

Let

- x0 = (y + i) mod 2^^8
- x1 = (floor(y / 2^^8) + i) mod 2^^8
- x2 = (floor(y / 2^^16) + i) mod 2^^8
- x3 = (floor(y / 2^^24) + i) mod 2^^8

Then

```
Rand[y, i, m] = (V0[x0] ^ V1[x1] ^ V2[x2] ^ V3[x3]) % m
```

5.3.5.2. Degree Generator

The degree generator Deg[v] is defined as follows, where v is a non-negative integer that is less than 2^^20 = 1048576. Given v, find index d in Table 1 such that f[d-1] <= v < f[d], and set Deg[v] = min(d, W-2).). Recall that W is derived from K' as described in Section 5.3.3.3.

Index d	f[d]	Index d	f[d]
0	0	1	5243
2	529531	3	704294
4	791675	5	844104
6	879057	7	904023
8	922747	9	937311
10	948962	11	958494
12	966438	13	973160
14	978921	15	983914
16	988283	17	992138
18	995565	19	998631
20	1001391	21	1003887
22	1006157	23	1008229
24	1010129	25	1011876
26	1013490	27	1014983
28	1016370	29	1017662
30	1048576		

Table 1: Defines the degree distribution for encoding symbols

5.3.5.3. Encoding Symbol Generator

The encoding symbol generator Enc[K', (C[0], C[1],..., C[L-1]), (d, a, b, d1, a1, b1)] takes the following inputs:

o K' is the number of source symbols for the extended source block. Let L, W, B, S, P and P1 be derived from K' as described in Section 5.3.3.3.

- (C[0], C[1],..., C[L-1]) is the array of L intermediate symbols (sub-symbols) generated as described in Section 5.3.3.4
- (d, a, b, d1, a1, b1) is a source tuple determined from ISI X using the Tuple generator defined in Section 5.3.5.4, whereby
    * d is a positive integer denoting an encoding symbol LT degree
    * a is a positive integer between 1 and W-1 inclusive
    * b is a non-negative integer between 0 and W-1 inclusive
    * d1 is a positive integer that has value either 2 or 3 inclusive denoting an encoding symbol PI degree
    * a1 is a positive integer between 1 and P1-1 inclusive
    * b1 is a non-negative integer between 0 and P1-1 inclusive

The encoding symbol generator produces a single encoding symbol as output (referred to as result), according to the following algorithm:

```
o  result = C[b]

o  For j = 1, ..., d-1 do

   *  b = (b + a) % W

   *  result = result + C[b]

o  While (b1 >= P) do b1 = (b1+a1) % P1

o  result = result + C[W+b1]

o  For j = 1, ..., d1-1 do

   *  b1 = (b1 + a1) % P1

   *  While (b1 >= P) do b1 = (b1+a1) % P1

   *  result = result + C[W+b1]

o  Return result
```

5.3.5.4. Tuple generator

The tuple generator Tuple[K',X] takes the following inputs:

o K' - The number of source symbols in the extended source block

o X - An ISI

Let

o L be determined from K' as described in Section 5.3.3.3

o J=J(K') be the systematic index associated with K', as defined in Table 2 inSection 5.6

The output of tuple generator is a tuple, (d, a, b, d1, a1, b1), determined as follows:

```
o  A = 53591 + J*997

o  if (A % 2 == 0) { A = A + 1 }

o  B = 10267*(J+1)

o  y = (B + X*A) % 2^^32

o  v = Rand[y, 0, 2^^20]

o  d = Deg[v]

o  a = 1 + Rand[y, 1, W-1]

o  b = Rand[y, 2, W]

o  If (d<4) { d1 = 2 + Rand[X, 3, 2] } else { d1 = 2 }

o  a1 = 1 + Rand[X, 4, P1-1]

o  b1 = Rand[X, 5, P1]
```

5.4. Example FEC decoder

5.4.1. General

This section describes an efficient decoding algorithm for the RaptorQ code introduced in this specification. Note that each received encoding symbol is a known linear combination of the intermediate symbols. So each received encoding symbol provides a linear equation among the intermediate symbols, which, together with the known linear pre-coding relationships amongst the intermediate symbols gives a system of linear equations. Thus, any algorithm for solving systems of linear equations can successfully decode the intermediate symbols and hence the source symbols. However, the algorithm chosen has a major effect on the computational efficiency of the decoding.

5.4.2. Decoding an extended source block

5.4.2.1. General

It is assumed that the decoder knows the structure of the source block it is to decode, including the symbol size, T, and the number K of symbols in the source block and the number K' of source symbols in the extended source block.

From the algorithms described in Sections Section 5.3, the RaptorQ decoder can calculate the total number L = K'+S+H of intermediate symbols and determine how they were generated from the extended source block to be decoded. In this description it is assumed that the received encoding symbols for the extended source block to be decoded are passed to the decoder. Furthermore, for each such encoding symbol it is assumed that the number and set of intermediate symbols whose sum is equal to the encoding symbol is passed to the decoder. In the case of source symbols, including padding symbols, the source symbol tuples described in Section 5.3.3.2 indicate the number and set of intermediate symbols which sum to give each source symbol.

Let N >= K' be the number of received encoding symbols to be used for decoding, including padding symbols for an extended source block and let M = S+H+N. Then with the notation of Section 5.3.3.4.2 we have A*C=D.

Decoding an extended source block is equivalent to decoding C from known A and D. It is clear that C can be decoded if and only if the rank of A is L. Once C has been decoded, missing source symbols can be obtained by using the source symbol tuples to determine the number and set of intermediate symbols which must be summed to obtain each missing source symbol.

The first step in decoding C is to form a decoding schedule. In this step A is converted, using Gaussian elimination (using row operations and row and column reorderings) and after discarding M - L rows, into the L by L identity matrix. The decoding schedule consists of the sequence of row operations and row and column re-orderings during the Gaussian elimination process, and only depends on A and not on D. The decoding of C from D can take place concurrently with the forming of the decoding schedule, or the decoding can take place afterwards based on the decoding schedule.

The correspondence between the decoding schedule and the decoding of C is as follows. Let c[0] = 0, c[1] = 1, ..., c[L-1] = L-1 and d[0] = 0, d[1] = 1, ..., d[M-1] = M-1 initially.

o Each time a multiple, beta, of row i of A is added to row i' in the decoding schedule then in the decoding process the symbol beta*D[d[i]] is added to symbol D[d[i']] .

o Each time a row i of A is multiplied by an octet beta, then in the decoding process the symbol D[d[i]] is also multiplied by beta.

o Each time row i is exchanged with row i' in the decoding schedule then in the decoding process the value of d[i] is exchanged with the value of d[i'].

o Each time column j is exchanged with column j' in the decoding schedule then in the decoding process the value of c[j] is exchanged with the value of c[j'].

From this correspondence it is clear that the total number of operations on symbols in the decoding of the extended source block is the number of row operations (not exchanges) in the Gaussian elimination. Since A is the L by L identity matrix after the Gaussian elimination and after discarding the last M - L rows, it is clear at the end of successful decoding that the L symbols D[d[0]], D[d[1]],..., D[d[L-1]] are the values of the L symbols C[c[0]], C[c[1]],..., C[c[L-1]].

The order in which Gaussian elimination is performed to form the decoding schedule has no bearing on whether or not the decoding is successful. However, the speed of the decoding depends heavily on the order in which Gaussian elimination is performed. (Furthermore, maintaining a sparse representation of A is crucial, although this is not described here). The remainder of this section describes an order in which Gaussian elimination could be performed that is relatively efficient.

5.4.2.2. First Phase

In the first phase of the Gaussian elimination the matrix A is conceptually partitioned into submatrices and additionally, a matrix X is created. This matrix has as many rows and columns as A, and it will be a lower triangular matrix throughout the first phase. At the beginning of this phase, the matrix A is copied into the matrix X.

The submatrix sizes are parameterized by non-negative integers i and u which are initialized to 0 and P, the number of PI symbols, respectively. The submatrices of A are:

1. The submatrix I defined by the intersection of the first i rows and first i columns. This is the identity matrix at the end of each step in the phase.

2. The submatrix defined by the intersection of the first i rows and all but the first i columns and last u columns. All entries of this submatrix are zero.

3. The submatrix defined by the intersection of the first i columns and all but the first i rows. All entries of this submatrix are zero.

4. The submatrix U defined by the intersection of all the rows and the last u columns.

5. The submatrix V formed by the intersection of all but the first i columns and the last u columns and all but the first i rows.

Figure 6 illustrates the submatrices of A. At the beginning of the first phase V = A. In each step, a row of A is chosen.

```
+-----------+-----------------+---------+
|           |                 |         |
|     I     |    All Zeros    |         |
|           |                 |         |
+-----------+-----------------+    U    |
|           |                 |         |
|           |                 |         |
| All Zeros |       V         |         |
|           |                 |         |
|           |                 |         |
+-----------+-----------------+---------+
```

Figure 6: Submatrices of A in the first phase

The following graph defined by the structure of V is used in determining which row of A is chosen. The columns that intersect V are the nodes in the graph, and the rows that have exactly 2 non-zero entries in V and are not HDPC rows are the edges of the graph that connect the two columns (nodes) in the positions of the two ones. A component in this graph is a maximal set of nodes (columns) and edges (rows) such that there is a path between each pair of nodes/edges in the graph. The size of a component is the number of nodes (columns) in the component.

There are at most L steps in the first phase. The phase ends successfully when i + u = L, i.e., when V and the all zeroes submatrix above V have disappeared and A consists of I, the all zeroes submatrix below I, and U. The phase ends unsuccessfully in decoding failure if at some step before V disappears there is no non-zero row in V to choose in that step. In each step, a row of A is chosen as follows:

o If all entries of V are zero then no row is chosen and decoding fails.

o Let r be the minimum integer such that at least one row of A has exactly r ones in V.

  * If r != 2 then choose a row with exactly r ones in V with minimum original degree among all such rows, except that HDPC rows should not be chosen until all non-HDPC rows have been processed.

  * If r = 2 then choose any row with exactly 2 ones in V that is part of a maximum size component in the graph described above which is defined by V.

After the row is chosen in this step the first row of A that intersects V is exchanged with the chosen row so that the chosen row is the first row that intersects V. The columns of A among those that intersect V are reordered so that one of the r ones in the chosen row appears in the first column of V and so that the remaining r-1 ones appear in the last columns of V. The same row and column operations are also performed on the matrix X. Then, an appropriate multiple of the chosen row is added to all the other rows of A below the chosen row that have a non-zero entry in the first column of V. Specifically, if a row below the chosen row has entry beta in the first column of V, and the chosen row has entry alpha in the first column of V, then beta/alpha multiplied by the chosen row is added to this row to leave a zero value in the first column of V. Finally, i is incremented by 1 and u is incremented by r-1, which completes the step.

Note that efficiency can be improved if the row operations identified above are not actually performed until the affected row is itself chosen during the decoding process. This avoids processing of row operations for rows which are not eventually used in the decoding process and in particular avoid those rows for which beta!=1 until they are actually required. Furthermore, the row operations required for the HDPC rows may be performed for all such rows in one process, by using the algorithm described in Section 5.3.3.3.

5.4.2.3. Second Phase

At this point, all the entries of X outside the first i rows and i columns are discarded, so that X has lower triangular form. The last i rows and columns of X are discarded, so that X now has i rows i columns. The submatrix U is further partitioned into the first i rows, U_upper, and the remaining M - i rows, U_lower. Gaussian elimination is performed in the second phase on U_lower to either determine that its rank is less than u (decoding failure) or to convert it into a matrix where the first u rows is the identity matrix (success of the second phase). Call this u by u identity matrix I_u. The M - L rows of A that intersect U_lower - I_u are discarded. After this phase A has L rows and L columns.

5.4.2.4. Third Phase

After the second phase the only portion of A which needs to be zeroed out to finish converting A into the L by L identity matrix is U_upper. The number of rows i of the submatrix U_upper is generally much larger than the number of columns u of U_upper. Moreover, at this time, the matrix U_upper is typically dense, i.e., the number of nonzero entries of this matrix is large. To reduce this matrix to a sparse form, the sequence of operations performed to obtain the matrix U_lower needs to be inverted. To this end, the matrix X is multiplied with the submatrix of A consisting of the first i rows of A. After this operation the submatrix of A consisting of the intersection of the first i rows and columns equals to X, whereas the matrix U_upper is transformed to a sparse form.

5.4.2.5. Fourth Phase

For each of the first i rows of U_upper do the following: if the row has a nonzero entry at position j, and if the value of that nonzero entry is b, then add to this row b times row j of I_u. After this step, the submatrix of A consisting of the intersection of the first i rows and columns is equal to X, the submatrix U_upper consists of zeros, the submatrix consisting of the intersection of the last u rows and the first i columns consists of zeros, and the submatrix consisting of the last u rows and columns is is the matrix I_u.

5.4.2.6. Fifth Phase

For j from 1 to i perform the following operations:

1. if A[j,j] is not one, then divide row j of A by A[j,j].

2. For l from 1 to j-1, if A[j,l] is nonzero, then add A[j,l] multiplied with row l of A to row j of A.

After this phase A is the L by L identity matrix and a complete decoding schedule has been successfully formed. Then, the corresponding decoding consisting of summing known encoding symbols can be executed to recover the intermediate symbols based on the decoding schedule. The tuples associated with all source symbols are computed according to Section 5.3.3.2. The tuples for received source symbols are used in the decoding. The tuples for missing source symbols are used to determine which intermediate symbols need to be summed to recover the missing source symbols.

5.5. Random Numbers

The four arrays V0, V1, V2 and V3 used in Section 5.3.5.1 are provided below. There are 256 entries in each of the four arrays. The indexing into each array starts at 0, and the entries are 32-bit unsigned integers.

5.5.1. The table V0

```
251291136, 3952231631, 3370958628, 4070167936, 123631495, 3351110283,
3218676425, 2011642291, 774603218, 2402805061, 1004366930,
1843948209, 428891132, 3746331984, 1591258008, 3067016507,
1433388735, 504005498, 2032657933, 3419319784, 2805686246,
3102436986, 3808671154, 2501582075, 3978944421, 246043949,
4016898363, 649743608, 1974987508, 2651273766, 2357956801, 689605112,
715807172, 2722736134, 191939188, 3535520147, 3277019569, 1470435941,
3763101702, 3232409631, 122701163, 3920852693, 782246947, 372121310,
2995604341, 2045698575, 2332962102, 4005368743, 218596347,
3415381967, 4207612806, 861117671, 3676575285, 2581671944,
3312220480, 681232419, 307306866, 4112503940, 1158111502, 709227802,
2724140433, 4201101115, 4215970289, 4048876515, 3031661061,
1909085522, 510985033, 1361682810, 129243379, 3142379587, 2569842483,
3033268270, 1658118006, 932109358, 1982290045, 2983082771,
3007670818, 3448104768, 683749698, 778296777, 1399125101, 1939403708,
1692176003, 3868299200, 1422476658, 593093658, 1878973865,
2526292949, 1591602827, 3986158854, 3964389521, 2695031039,
1942050155, 424618399, 1347204291, 2669179716, 2434425874,
2540801947, 1384069776, 4123580443, 1523670218, 2708475297,
1046771089, 2229796016, 1255426612, 4213663089, 1521339547,
3041843489, 420130494, 10677091, 515623176, 3457502702, 2115821274,
2720124766, 3242576090, 854310108, 425973987, 325832382, 1796851292,
2462744411, 1976681690, 1408671665, 1228817808, 3917210003,
263976645, 2593736473, 2471651269, 4291353919, 650792940, 1191583883,
3046561335, 2466530435, 2545983082, 969168436, 2019348792,
2268075521, 1169345068, 3250240009, 3963499681, 2560755113,
911182396, 760842409, 3569308693, 2687243553, 381854665, 2613828404,
2761078866, 1456668111, 883760091, 3294951678, 1604598575,
1985308198, 1014570543, 2724959607, 3062518035, 3115293053,
   138853680, 4160398285, 3322241130, 2068983570, 2247491078,
   3669524410, 1575146607, 828029864, 3732001371, 3422026452,
   3370954177, 4006626915, 543812220, 1243116171, 3928372514,
   2791443445, 4081325272, 2280435605, 885616073, 616452097, 3188863436,
   2780382310, 2340014831, 1208439576, 258356309, 3837963200,
   2075009450, 3214181212, 3303882142, 880813252, 1355575717, 207231484,
   2420803184, 358923368, 1617557768, 3272161958, 1771154147,
   2842106362, 1751209208, 1421030790, 658316681, 194065839, 3241510581,
   38625260, 301875395, 4176141739, 297312930, 2137802113, 1502984205,
   3669376622, 3728477036, 234652930, 2213589897, 2734638932,
   1129721478, 3187422815, 2859178611, 3284308411, 3819792700,
   3557526733, 451874476, 1740576081, 3592838701, 1709429513,
   3702918379, 3533351328, 1641660745, 179350258, 2380520112,
   3936163904, 3685256204, 3156252216, 1854258901, 2861641019,
   3176611298, 834787554, 331353807, 517858103, 3010168884, 4012642001,
   2217188075, 3756943137, 3077882590, 2054995199, 3081443129,
   3895398812, 1141097543, 2376261053, 2626898255, 2554703076,
   401233789, 1460049922, 678083952, 1064990737, 940909784, 1673396780,
   528881783, 1712547446, 3629685652, 1358307511
```

5.5.2. The table V1

```
   807385413, 2043073223, 3336749796, 1302105833, 2278607931, 541015020,
   1684564270, 372709334, 3508252125, 1768346005, 1270451292,
   2603029534, 2049387273, 3891424859, 2152948345, 4114760273,
   915180310, 3754787998, 700503826, 2131559305, 1308908630, 224437350,
   4065424007, 3638665944, 1679385496, 3431345226, 1779595665,
   3068494238, 1424062773, 1033448464, 4050396853, 3302235057,
   420600373, 2868446243, 311689386, 259047959, 4057180909, 1575367248,
   4151214153, 110249784, 3006865921, 4293710613, 3501256572, 998007483,
   499288295, 1205710710, 2997199489, 640417429, 3044194711, 486690751,
   2686640734, 2394526209, 2521660077, 49993987, 3843885867, 4201106668,
   415906198, 19296841, 2402488407, 2137119134, 1744097284, 579965637,
   2037662632, 852173610, 2681403713, 1047144830, 2982173936, 910285038,
   4187576520, 2589870048, 989448887, 3292758024, 506322719, 176010738,
   1865471968, 2619324712, 564829442, 1996870325, 339697593, 4071072948,
   3618966336, 2111320126, 1093955153, 957978696, 892010560, 1854601078,
   1873407527, 2498544695, 2694156259, 1927339682, 1650555729,
   183933047, 3061444337, 2067387204, 228962564, 3904109414, 1595995433,
   1780701372, 2463145963, 307281463, 3237929991, 3852995239,
   2398693510, 3754138664, 522074127, 146352474, 4104915256, 3029415884,
   3545667983, 332038910, 976628269, 3123492423, 3041418372, 2258059298,
   2139377204, 3243642973, 3226247917, 3674004636, 2698992189,
   3453843574, 1963216666, 3509855005, 2358481858, 747331248,
   1957348676, 1097574450, 2435697214, 3870972145, 1888833893,
   2914085525, 4161315584, 1273113343, 3269644828, 3681293816,
   412536684, 1156034077, 3823026442, 1066971017, 3598330293,
   1979273937, 2079029895, 1195045909, 1071986421, 2712821515,
   3377754595, 2184151095, 750918864, 2585729879, 4249895712,
   1832579367, 1192240192, 946734366, 31230688, 3174399083, 3549375728,
   1642430184, 1904857554, 861877404, 3277825584, 4267074718,
   3122860549, 666423581, 644189126, 226475395, 307789415, 1196105631,
   3191691839, 782852669, 1608507813, 1847685900, 4069766876,
   3931548641, 2526471011, 766865139, 2115084288, 4259411376,
   3323683436, 568512177, 3736601419, 1800276898, 4012458395, 1823982,
   27980198, 2023839966, 869505096, 431161506, 1024804023, 1853869307,
   3393537983, 1500703614, 3019471560, 1351086955, 3096933631,
   3034634988, 2544598006, 1230942551, 3362230798, 159984793, 491590373,
   3993872886, 3681855622, 903593547, 3535062472, 1799803217, 772984149,
   895863112, 1899036275, 4187322100, 101856048, 234650315, 3183125617,
   3190039692, 525584357, 1286834489, 455810374, 1869181575, 922673938,
   3877430102, 3422391938, 1414347295, 1971054608, 3061798054,
   830555096, 2822905141, 167033190, 1079139428, 4210126723, 3593797804,
   429192890, 372093950, 1779187770, 3312189287, 204349348, 452421568,
   2800540462, 3733109044, 1235082423, 1765319556, 3174729780,
   3762994475, 3171962488, 442160826, 198349622, 45942637, 1324086311,
   2901868599, 678860040, 3812229107, 19936821, 1119590141, 3640121682,
   3545931032, 2102949142, 2828208598, 3603378023, 4135048896
```

5.5.3. The table V2

```
   1629829892, 282540176, 2794583710, 496504798, 2990494426, 3070701851,
   2575963183, 4094823972, 2775723650, 4079480416, 176028725,
   2246241423, 3732217647, 2196843075, 1306949278, 4170992780,
   4039345809, 3209664269, 3387499533, 293063229, 3660290503,
   2648440860, 2531406539, 3537879412, 773374739, 4184691853,
   1804207821, 3347126643, 3479377103, 3970515774, 1891731298,
   2368003842, 3537588307, 2969158410, 4230745262, 831906319,
   2935838131, 264029468, 120852739, 3200326460, 355445271, 2296305141,
   1566296040, 1760127056, 20073893, 3427103620, 2866979760, 2359075957,
   2025314291, 1725696734, 3346087406, 2690756527, 99815156, 4248519977,
   2253762642, 3274144518, 598024568, 3299672435, 556579346, 4121041856,
   2896948975, 3620123492, 918453629, 3249461198, 2231414958,
   3803272287, 3657597946, 2588911389, 242262274, 1725007475,
   2026427718, 46776484, 2873281403, 2919275846, 3177933051, 1918859160,
   2517854537, 1857818511, 3234262050, 479353687, 200201308, 2801945841,
   1621715769, 483977159, 423502325, 3689396064, 1850168397, 3359959416,
   3459831930, 841488699, 3570506095, 930267420, 1564520841, 2505122797,
   593824107, 1116572080, 819179184, 3139123629, 1414339336, 1076360795,
   512403845, 177759256, 1701060666, 2239736419, 515179302, 2935012727,
   3821357612, 1376520851, 2700745271, 966853647, 1041862223, 715860553,
   171592961, 1607044257, 1227236688, 3647136358, 1417559141,
   4087067551, 2241705880, 4194136288, 1439041934, 20464430, 119668151,
   2021257232, 2551262694, 1381539058, 4082839035, 498179069, 311508499,
   3580908637, 2889149671, 142719814, 1232184754, 3356662582,
   2973775623, 1469897084, 1728205304, 1415793613, 50111003, 3133413359,
   4074115275, 2710540611, 2700083070, 2457757663, 2612845330,
   3775943755, 2469309260, 2560142753, 3020996369, 1691667711,
   4219602776, 1687672168, 1017921622, 2307642321, 368711460,
   3282925988, 213208029, 4150757489, 3443211944, 2846101972,
   4106826684, 4272438675, 2199416468, 3710621281, 497564971, 285138276,
   765042313, 916220877, 3402623607, 2768784621, 1722849097, 3386397442,
   487920061, 3569027007, 3424544196, 217781973, 2356938519, 3252429414,
   145109750, 2692588106, 2454747135, 1299493354, 4120241887,
   2088917094, 932304329, 1442609203, 952586974, 3509186750, 753369054,
   854421006, 1954046388, 2708927882, 4047539230, 3048925996,
   1667505809, 805166441, 1182069088, 4265546268, 4215029527,
   3374748959, 373532666, 2454243090, 2371530493, 3651087521,
   2619878153, 1651809518, 1553646893, 1227452842, 703887512,
   3696674163, 2552507603, 2635912901, 895130484, 3287782244,
   3098973502, 990078774, 3780326506, 2290845203, 41729428, 1949580860,
   2283959805, 1036946170, 1694887523, 4880696, 466000198, 2765355283,
   3318686998, 1266458025, 3919578154, 3545413527, 2627009988,
   3744680394, 1696890173, 3250684705, 4142417708, 915739411,
   3308488877, 1289361460, 2942552331, 1169105979, 3342228712,
   698560958, 1356041230, 2401944293, 107705232, 3701895363, 903928723,
   3646581385, 844950914, 1944371367, 3863894844, 2946773319,
   1972431613, 1706989237, 29917467, 3497665928
```

5.5.4. The table V3

```
   1191369816, 744902811, 2539772235, 3213192037, 3286061266,
   1200571165, 2463281260, 754888894, 714651270, 1968220972, 3628497775,
   1277626456, 1493398934, 364289757, 2055487592, 3913468088,
   2930259465, 902504567, 3967050355, 2056499403, 692132390, 186386657,
   832834706, 859795816, 1283120926, 2253183716, 3003475205, 1755803552,
   2239315142, 4271056352, 2184848469, 769228092, 1249230754,
   1193269205, 2660094102, 642979613, 1687087994, 2726106182, 446402913,
   4122186606, 3771347282, 37667136, 192775425, 3578702187, 1952659096,
   3989584400, 3069013882, 2900516158, 4045316336, 3057163251,
   1702104819, 4116613420, 3575472384, 2674023117, 1409126723,
   3215095429, 1430726429, 2544497368, 1029565676, 1855801827,
   4262184627, 1854326881, 2906728593, 3277836557, 2787697002,
   2787333385, 3105430738, 2477073192, 748038573, 1088396515,
   1611204853, 201964005, 3745818380, 3654683549, 3816120877,
   3915783622, 2563198722, 1181149055, 33158084, 3723047845, 3790270906,
   3832415204, 2959617497, 372900708, 1286738499, 1932439099,
   3677748309, 2454711182, 2757856469, 2134027055, 2780052465,
   3190347618, 3758510138, 3626329451, 1120743107, 1623585693,
   1389834102, 2719230375, 3038609003, 462617590, 260254189, 3706349764,
   2556762744, 2874272296, 2502399286, 4216263978, 2683431180,
   2168560535, 3561507175, 668095726, 680412330, 3726693946, 4180630637,
   3335170953, 942140968, 2711851085, 2059233412, 4265696278,
   3204373534, 232855056, 881788313, 2258252172, 2043595984, 3758795150,
   3615341325, 2138837681, 1351208537, 2923692473, 3402482785,
   2105383425, 2346772751, 499245323, 3417846006, 2366116814,
   2543090583, 1828551634, 3148696244, 3853884867, 1364737681,
   2200687771, 2689775688, 232720625, 4071657318, 2671968983,
   3531415031, 1212852141, 867923311, 3740109711, 1923146533,
   3237071777, 3100729255, 3247856816, 906742566, 4047640575,
   4007211572, 3495700105, 1171285262, 2835682655, 1634301229,
   3115169925, 2289874706, 2252450179, 944880097, 371933491, 1649074501,
   2208617414, 2524305981, 2496569844, 2667037160, 1257550794,
   3399219045, 3194894295, 1643249887, 342911473, 891025733, 3146861835,
   3789181526, 938847812, 1854580183, 2112653794, 2960702988,
   1238603378, 2205280635, 1666784014, 2520274614, 3355493726,
   2310872278, 3153920489, 2745882591, 1200203158, 3033612415,
   2311650167, 1048129133, 4206710184, 4209176741, 2640950279,
   2096382177, 4116899089, 3631017851, 4104488173, 1857650503,
   3801102932, 445806934, 3055654640, 897898279, 3234007399, 1325494930,
   2982247189, 1619020475, 2720040856, 885096170, 3485255499,
   2983202469, 3891011124, 546522756, 1524439205, 2644317889,
   2170076800, 2969618716, 961183518, 1081831074, 1037015347,
   3289016286, 2331748669, 620887395, 303042654, 3990027945, 1562756376,
   3413341792, 2059647769, 2823844432, 674595301, 2457639984,
   4076754716, 2447737904, 1583323324, 625627134, 3076006391, 345777990,
   1684954145, 879227329, 3436182180, 1522273219, 3802543817,
   1456017040, 1897819847, 2970081129, 1382576028, 3820044861,
   1044428167, 612252599, 3340478395, 2150613904, 3397625662,
   3573635640, 3432275192
```

5.6. Systematic indices and other parameters

Table 2 below specifies the supported values of K'. The table also specifies for each supported value of K' the systematic index J(K'), the number H(K') of HDPC symbols, the number S(K') of LDPC symbols, and the number W(K') of LT symbols. For each value of K', the corresponding values of S(K') and W(K') are prime numbers.

The systematic index J(K') is designed to have the property that the set of source symbol tuples (d[0], a[0], b[0], d1[0], a1[0], b1[0]), ..., (d[K'-1], a[K'-1], b[K'-1], d1[K'-1], a1[K'-1], b1[K'-1]) are such that the L intermediate symbols are uniquely defined, i.e., the matrix A in Figure 6 has full rank and is therefore invertible.

K'	J(K')	S(K')	H(K')	W(K')
10	254	7	10	17
12	630	7	10	19

18	682	11	10	29
20	293	11	10	31
26	80	11	10	37
30	566	11	10	41
32	860	11	10	43
36	267	11	10	47
42	822	11	10	53
46	506	13	10	59
48	589	13	10	61
49	87	13	10	61
55	520	13	10	67
60	159	13	10	71
62	235	13	10	73
69	157	13	10	79
75	502	17	10	89
84	334	17	10	97
88	583	17	10	101
91	66	17	10	103
95	352	17	10	107
97	365	17	10	109
101	562	17	10	113
114	5	19	10	127
119	603	19	10	131
125	721	19	10	137

127	28	19	10	139
138	660	19	10	149
140	829	19	10	151
149	900	23	10	163
153	930	23	10	167
160	814	23	10	173
166	661	23	10	179
168	693	23	10	181
179	780	23	10	191
181	605	23	10	193
185	551	23	10	197
187	777	23	10	199
200	491	23	10	211
213	396	23	10	223
217	764	29	10	233
225	843	29	10	241
236	646	29	10	251
242	557	29	10	257
248	608	29	10	263
257	265	29	10	271
263	505	29	10	277
269	722	29	10	283
280	263	29	10	293
295	999	29	10	307

301	874	29	10	313
305	160	29	10	317
324	575	31	10	337
337	210	31	10	349
341	513	31	10	353
347	503	31	10	359
355	558	31	10	367
362	932	31	10	373
368	404	31	10	379
372	520	37	10	389
380	846	37	10	397
385	485	37	10	401
393	728	37	10	409
405	554	37	10	421
418	471	37	10	433
428	641	37	10	443
434	732	37	10	449
447	193	37	10	461
453	934	37	10	467
466	864	37	10	479
478	790	37	10	491
486	912	37	10	499
491	617	37	10	503
497	587	37	10	509

511	800	37	10	523
526	923	41	10	541
532	998	41	10	547
542	92	41	10	557
549	497	41	10	563
557	559	41	10	571
563	667	41	10	577
573	912	41	10	587
580	262	41	10	593
588	152	41	10	601
594	526	41	10	607
600	268	41	10	613
606	212	41	10	619
619	45	41	10	631
633	898	43	10	647
640	527	43	10	653
648	558	43	10	661
666	460	47	10	683
675	5	47	10	691
685	895	47	10	701
693	996	47	10	709
703	282	47	10	719
718	513	47	10	733
728	865	47	10	743

736	870	47	10	751
747	239	47	10	761
759	452	47	10	773
778	862	53	10	797
792	852	53	10	811
802	643	53	10	821
811	543	53	10	829
821	447	53	10	839
835	321	53	10	853
845	287	53	10	863
860	12	53	10	877
870	251	53	10	887
891	30	53	10	907
903	621	53	10	919
913	555	53	10	929
926	127	53	10	941
938	400	53	10	953
950	91	59	10	971
963	916	59	10	983
977	935	59	10	997
989	691	59	10	1009
1002	299	59	10	1021
1020	282	59	10	1039
1032	824	59	10	1051

1050	536	59	11	1069
1074	596	59	11	1093
1085	28	59	11	1103
1099	947	59	11	1117
1111	162	59	11	1129
1136	536	59	11	1153
1152	1000	61	11	1171
1169	251	61	11	1187
1183	673	61	11	1201
1205	559	61	11	1223
1220	923	61	11	1237
1236	81	67	11	1259
1255	478	67	11	1277
1269	198	67	11	1291
1285	137	67	11	1307
1306	75	67	11	1327
1347	29	67	11	1367
1361	231	67	11	1381
1389	532	67	11	1409
1404	58	67	11	1423
1420	60	67	11	1439
1436	964	71	11	1459
1461	624	71	11	1483
1477	502	71	11	1499

1502	636	71	11	1523
1522	986	71	11	1543
1539	950	71	11	1559
1561	735	73	11	1583
1579	866	73	11	1601
1600	203	73	11	1621
1616	83	73	11	1637
1649	14	73	11	1669
1673	522	79	11	1699
1698	226	79	11	1723
1716	282	79	11	1741
1734	88	79	11	1759
1759	636	79	11	1783
1777	860	79	11	1801
1800	324	79	11	1823
1824	424	79	11	1847
1844	999	79	11	1867
1863	682	83	11	1889
1887	814	83	11	1913
1906	979	83	11	1931
1926	538	83	11	1951
1954	278	83	11	1979
1979	580	83	11	2003
2005	773	83	11	2029

2040	911	89	11	2069
2070	506	89	11	2099
2103	628	89	11	2131
2125	282	89	11	2153
2152	309	89	11	2179
2195	858	89	11	2221
2217	442	89	11	2243
2247	654	89	11	2273
2278	82	97	11	2311
2315	428	97	11	2347
2339	442	97	11	2371
2367	283	97	11	2399
2392	538	97	11	2423
2416	189	97	11	2447
2447	438	97	11	2477
2473	912	97	11	2503
2502	1	97	11	2531
2528	167	97	11	2557
2565	272	97	11	2593
2601	209	101	11	2633
2640	927	101	11	2671
2668	386	101	11	2699
2701	653	101	11	2731
2737	669	101	11	2767

2772	431	101	11	2801
2802	793	103	11	2833
2831	588	103	11	2861
2875	777	107	11	2909
2906	939	107	11	2939
2938	864	107	11	2971
2979	627	107	11	3011
3015	265	109	11	3049
3056	976	109	11	3089
3101	988	113	11	3137
3151	507	113	11	3187
3186	640	113	11	3221
3224	15	113	11	3259
3265	667	113	11	3299
3299	24	127	11	3347
3344	877	127	11	3391
3387	240	127	11	3433
3423	720	127	11	3469
3466	93	127	11	3511
3502	919	127	11	3547
3539	635	127	11	3583
3579	174	127	11	3623
3616	647	127	11	3659
3658	820	127	11	3701

3697	56	127	11	3739
3751	485	127	11	3793
3792	210	127	11	3833
3840	124	127	11	3881
3883	546	127	11	3923
3924	954	131	11	3967
3970	262	131	11	4013
4015	927	131	11	4057
4069	957	131	11	4111
4112	726	137	11	4159
4165	583	137	11	4211
4207	782	137	11	4253
4252	37	137	11	4297
4318	758	137	11	4363
4365	777	137	11	4409
4418	104	139	11	4463
4468	476	139	11	4513
4513	113	149	11	4567
4567	313	149	11	4621
4626	102	149	11	4679
4681	501	149	11	4733
4731	332	149	11	4783
4780	786	149	11	4831
4838	99	149	11	4889

4901	658	149	11	4951
4954	794	149	11	5003
5008	37	151	11	5059
5063	471	151	11	5113
5116	94	157	11	5171
5172	873	157	11	5227
5225	918	157	11	5279
5279	945	157	11	5333
5334	211	157	11	5387
5391	341	157	11	5443
5449	11	163	11	5507
5506	578	163	11	5563
5566	494	163	11	5623
5637	694	163	11	5693
5694	252	163	11	5749
5763	451	167	11	5821
5823	83	167	11	5881
5896	689	167	11	5953
5975	488	173	11	6037
6039	214	173	11	6101
6102	17	173	11	6163
6169	469	173	11	6229
6233	263	179	11	6299
6296	309	179	11	6361

6363	984	179	11	6427
6427	123	179	11	6491
6518	360	179	11	6581
6589	863	181	11	6653
6655	122	181	11	6719
6730	522	191	11	6803
6799	539	191	11	6871
6878	181	191	11	6949
6956	64	191	11	7027
7033	387	191	11	7103
7108	967	191	11	7177
7185	843	191	11	7253
7281	999	193	11	7351
7360	76	197	11	7433
7445	142	197	11	7517
7520	599	197	11	7591
7596	576	199	11	7669
7675	176	211	11	7759
7770	392	211	11	7853
7855	332	211	11	7937
7935	291	211	11	8017
8030	913	211	11	8111
8111	608	211	11	8191
8194	212	211	11	8273

8290	696	211	11	8369
8377	931	223	11	8467
8474	326	223	11	8563
8559	228	223	11	8647
8654	706	223	11	8741
8744	144	223	11	8831
8837	83	223	11	8923
8928	743	223	11	9013
9019	187	223	11	9103
9111	654	227	11	9199
9206	359	227	11	9293
9303	493	229	11	9391
9400	369	233	11	9491
9497	981	233	11	9587
9601	276	239	11	9697
9708	647	239	11	9803
9813	389	239	11	9907
9916	80	239	11	10009
10017	396	241	11	10111
10120	580	251	11	10223
10241	873	251	11	10343
10351	15	251	11	10453
10458	976	251	11	10559
10567	584	251	11	10667

10676	267	257	11	10781
10787	876	257	11	10891
10899	642	257	12	11003
11015	794	257	12	11119
11130	78	263	12	11239
11245	736	263	12	11353
11358	882	269	12	11471
11475	251	269	12	11587
11590	434	269	12	11701
11711	204	269	12	11821
11829	256	271	12	11941
11956	106	277	12	12073
12087	375	277	12	12203
12208	148	277	12	12323
12333	496	281	12	12451
12460	88	281	12	12577
12593	826	293	12	12721
12726	71	293	12	12853
12857	925	293	12	12983
13002	760	293	12	13127
13143	130	293	12	13267
13284	641	307	12	13421
13417	400	307	12	13553
13558	480	307	12	13693

13695	76	307	12	13829
13833	665	307	12	13967
13974	910	307	12	14107
14115	467	311	12	14251
14272	964	311	12	14407
14415	625	313	12	14551
14560	362	317	12	14699
14713	759	317	12	14851
14862	728	331	12	15013
15011	343	331	12	15161
15170	113	331	12	15319
15325	137	331	12	15473
15496	308	331	12	15643
15651	800	337	12	15803
15808	177	337	12	15959
15977	961	337	12	16127
16161	958	347	12	16319
16336	72	347	12	16493
16505	732	347	12	16661
16674	145	349	12	16831
16851	577	353	12	17011
17024	305	353	12	17183
17195	50	359	12	17359
17376	351	359	12	17539

17559	175	367	12	17729
17742	727	367	12	17911
17929	902	367	12	18097
18116	409	373	12	18289
18309	776	373	12	18481
18503	586	379	12	18679
18694	451	379	12	18869
18909	287	383	12	19087
19126	246	389	12	19309
19325	222	389	12	19507
19539	563	397	12	19727
19740	839	397	12	19927
19939	897	401	12	20129
20152	409	401	12	20341
20355	618	409	12	20551
20564	439	409	12	20759
20778	95	419	13	20983
20988	448	419	13	21191
21199	133	419	13	21401
21412	938	419	13	21613
21629	423	431	13	21841
21852	90	431	13	22063
22073	640	431	13	22283
22301	922	433	13	22511

22536	250	439	13	22751
22779	367	439	13	22993
23010	447	443	13	23227
23252	559	449	13	23473
23491	121	457	13	23719
23730	623	457	13	23957
23971	450	457	13	24197
24215	253	461	13	24443
24476	106	467	13	24709
24721	863	467	13	24953
24976	148	479	13	25219
25230	427	479	13	25471
25493	138	479	13	25733
25756	794	487	13	26003
26022	247	487	13	26267
26291	562	491	13	26539
26566	53	499	13	26821
26838	135	499	13	27091
27111	21	503	13	27367
27392	201	509	13	27653
27682	169	521	13	27953
27959	70	521	13	28229
28248	386	521	13	28517
28548	226	523	13	28817

28845	3	541	13	29131
29138	769	541	13	29423
29434	590	541	13	29717
29731	672	541	13	30013
30037	713	547	13	30323
30346	967	547	13	30631
30654	368	557	14	30949
30974	348	557	14	31267
31285	119	563	14	31583
31605	503	569	14	31907
31948	181	571	14	32251
32272	394	577	14	32579
32601	189	587	14	32917
32932	210	587	14	33247
33282	62	593	14	33601
33623	273	593	14	33941
33961	554	599	14	34283
34302	936	607	14	34631
34654	483	607	14	34981
35031	397	613	14	35363
35395	241	619	14	35731
35750	500	631	14	36097
36112	12	631	14	36457
36479	958	641	14	36833

36849	524	641	14	37201
37227	8	643	14	37579
37606	100	653	14	37967
37992	339	653	14	38351
38385	804	659	14	38749
38787	510	673	14	39163
39176	18	673	14	39551
39576	412	677	14	39953
39980	394	683	14	40361
40398	830	691	15	40787
40816	535	701	15	41213
41226	199	701	15	41621
41641	27	709	15	42043
42067	298	709	15	42467
42490	368	719	15	42899
42916	755	727	15	43331
43388	379	727	15	43801
43840	73	733	15	44257
44279	387	739	15	44701
44729	457	751	15	45161
45183	761	751	15	45613
45638	855	757	15	46073
46104	370	769	15	46549
46574	261	769	15	47017

47047	299	787	15	47507
47523	920	787	15	47981
48007	269	787	15	48463
48489	862	797	15	48953
48976	349	809	15	49451
49470	103	809	15	49943
49978	115	821	15	50461
50511	93	821	16	50993
51017	982	827	16	51503
51530	432	839	16	52027
52062	340	853	16	52571
52586	173	853	16	53093
53114	421	857	16	53623
53650	330	863	16	54163
54188	624	877	16	54713
54735	233	877	16	55259
55289	362	883	16	55817
55843	963	907	16	56393
56403	471	907	16	56951

Table 2: Systematic indices and other parameters

5.7. Operating with Octets, Symbols and Matrices

5.7.1. General

This remainder of this section describes the arithmetic operations that are used to generate encoding symbols from source symbols and to generate source symbols from encoding symbols. Mathematically, octets can be thought of as elements of a finite field, i.e., the finite field GF(256) with 256 elements, and thus the addition and multiplication operations and identity elements and inverses over both operations are defined. Matrix operations and symbol operations are defined based on the arithmetic operations on octets. This allows a full implementation of these arithmetic operations without having to understand the underlying mathematics of finite fields.

5.7.2. Arithmetic Operations on Octets

Octets are mapped to non-negative integers in the range 0 through 255 in the usual way: A single octet of data from a symbol, B[7],B[6],B[5],B[4],B[3],B[2],B[1],B[0], where B[7] is the highest order bit and B[0] is the lowest order bit, is mapped to the integer i=B[7]*128+B[6]*64+B[5]*32+B[4]*16+B[3]*8+B[2]*4+B[1]*2+B[0].

The addition of two octets u and v defined as the XOR operation, i.e.,

```
u + v = u ^ v.
```

Subtraction is defined in the same way, so we also have

```
u - v = u ^ v.
```

The zero element (additive identity) is the octet represented by the integer 0. The additive inverse of u is simply u, i.e.,

```
u + u = 0.
```

The multiplication of two octets is defined with the help of two tables OCT_EXP and OCT_LOG, which are given in Section 5.7.3 and Section 5.7.4, respectively. The table OCT_LOG maps octets (other than the zero element) to non-negative integers, and OCT_EXP maps non-negative integers to octets. For two octets u and v, we define

```
u * v =

    0, if either u or v are 0,

    OCT_EXP[OCT_LOG[u] + OCT_LOG[v]] otherwise.
```

Note that the '+' on the right hand side of the above is the usual integer addition, since its arguments are ordinary integers.

The division u / v of two octets u and v, and where v != 0, is defined as follows:

```
   u / v =

      0, if u == 0,

      OCT_EXP[OCT_LOG[u] - OCT_LOG[v] + 255] otherwise.
```

The one element (multiplicative identity) is the octet represented by the integer 1. For an octet u that is not the zero element, i.e., the multiplicative inverse of u is

```
   OCT_EXP[255 - OCT_LOG[u]].
```

The octet denoted by alpha is the octet with the integer representation 2. If i is a non-negative integer 0 <= i < 256, we have

```
   alpha^^i = OCT_EXP[i].
```

5.7.3. The table OCT_EXP

The table OCT_EXP contains 510 octets. The indexing starts at 0 and ranges up to 509, and the entries are the octets with the following positive integer representation:

```
1, 2, 4, 8, 16, 32, 64, 128, 29, 58, 116, 232, 205, 135, 19, 38, 76,
152, 45, 90, 180, 117, 234, 201, 143, 3, 6, 12, 24, 48, 96, 192, 157,
39, 78, 156, 37, 74, 148, 53, 106, 212, 181, 119, 238, 193, 159, 35,
70, 140, 5, 10, 20, 40, 80, 160, 93, 186, 105, 210, 185, 111, 222,
161, 95, 190, 97, 194, 153, 47, 94, 188, 101, 202, 137, 15, 30, 60,
120, 240, 253, 231, 211, 187, 107, 214, 177, 127, 254, 225, 223, 163,
91, 182, 113, 226, 217, 175, 67, 134, 17, 34, 68, 136, 13, 26, 52,
104, 208, 189, 103, 206, 129, 31, 62, 124, 248, 237, 199, 147, 59,
118, 236, 197, 151, 51, 102, 204, 133, 23, 46, 92, 184, 109, 218,
169, 79, 158, 33, 66, 132, 21, 42, 84, 168, 77, 154, 41, 82, 164, 85,
170, 73, 146, 57, 114, 228, 213, 183, 115, 230, 209, 191, 99, 198,
145, 63, 126, 252, 229, 215, 179, 123, 246, 241, 255, 227, 219, 171,
75, 150, 49, 98, 196, 149, 55, 110, 220, 165, 87, 174, 65, 130, 25,
50, 100, 200, 141, 7, 14, 28, 56, 112, 224, 221, 167, 83, 166, 81,
162, 89, 178, 121, 242, 249, 239, 195, 155, 43, 86, 172, 69, 138, 9,
18, 36, 72, 144, 61, 122, 244, 245, 247, 243, 251, 235, 203, 139, 11,
22, 44, 88, 176, 125, 250, 233, 207, 131, 27, 54, 108, 216, 173, 71,
142, 1, 2, 4, 8, 16, 32, 64, 128, 29, 58, 116, 232, 205, 135, 19, 38,
76, 152, 45, 90, 180, 117, 234, 201, 143, 3, 6, 12, 24, 48, 96, 192,
157, 39, 78, 156, 37, 74, 148, 53, 106, 212, 181, 119, 238, 193, 159,
35, 70, 140, 5, 10, 20, 40, 80, 160, 93, 186, 105, 210, 185, 111,
222, 161, 95, 190, 97, 194, 153, 47, 94, 188, 101, 202, 137, 15, 30,
60, 120, 240, 253, 231, 211, 187, 107, 214, 177, 127, 254, 225, 223,
163, 91, 182, 113, 226, 217, 175, 67, 134, 17, 34, 68, 136, 13, 26,
```

52, 104, 208, 189, 103, 206, 129, 31, 62, 124, 248, 237, 199, 147,
59, 118, 236, 197, 151, 51, 102, 204, 133, 23, 46, 92, 184, 109, 218,
169, 79, 158, 33, 66, 132, 21, 42, 84, 168, 77, 154, 41, 82, 164, 85,
170, 73, 146, 57, 114, 228, 213, 183, 115, 230, 209, 191, 99, 198,
145, 63, 126, 252, 229, 215, 179, 123, 246, 241, 255, 227, 219, 171,
75, 150, 49, 98, 196, 149, 55, 110, 220, 165, 87, 174, 65, 130, 25,
50, 100, 200, 141, 7, 14, 28, 56, 112, 224, 221, 167, 83, 166, 81,
162, 89, 178, 121, 242, 249, 239, 195, 155, 43, 86, 172, 69, 138, 9,
18, 36, 72, 144, 61, 122, 244, 245, 247, 243, 251, 235, 203, 139, 11,
22, 44, 88, 176, 125, 250, 233, 207, 131, 27, 54, 108, 216, 173, 71,
142

5.7.4. The table OCT_LOG

The table OCT_LOG contains 255 non-negative integers. The table is indexed by octets interpreted as integers. The octet corresponding to the zero element, which is represented by the integer 0, is excluded as an index, and thus indexing starts at 1 and ranges up to 255, and the entries are the following:

0, 1, 25, 2, 50, 26, 198, 3, 223, 51, 238, 27, 104, 199, 75, 4, 100,
224, 14, 52, 141, 239, 129, 28, 193, 105, 248, 200, 8, 76, 113, 5,
138, 101, 47, 225, 36, 15, 33, 53, 147, 142, 218, 240, 18, 130, 69,
29, 181, 194, 125, 106, 39, 249, 185, 201, 154, 9, 120, 77, 228, 114,
166, 6, 191, 139, 98, 102, 221, 48, 253, 226, 152, 37, 179, 16, 145,
34, 136, 54, 208, 148, 206, 143, 150, 219, 189, 241, 210, 19, 92,
131, 56, 70, 64, 30, 66, 182, 163, 195, 72, 126, 110, 107, 58, 40,
84, 250, 133, 186, 61, 202, 94, 155, 159, 10, 21, 121, 43, 78, 212,
229, 172, 115, 243, 167, 87, 7, 112, 192, 247, 140, 128, 99, 13, 103,
74, 222, 237, 49, 197, 254, 24, 227, 165, 153, 119, 38, 184, 180,
124, 17, 68, 146, 217, 35, 32, 137, 46, 55, 63, 209, 91, 149, 188,
207, 205, 144, 135, 151, 178, 220, 252, 190, 97, 242, 86, 211, 171,
20, 42, 93, 158, 132, 60, 57, 83, 71, 109, 65, 162, 31, 45, 67, 216,
183, 123, 164, 118, 196, 23, 73, 236, 127, 12, 111, 246, 108, 161,
59, 82, 41, 157, 85, 170, 251, 96, 134, 177, 187, 204, 62, 90, 203,
89, 95, 176, 156, 169, 160, 81, 11, 245, 22, 235, 122, 117, 44, 215,
79, 174, 213, 233, 230, 231, 173, 232, 116, 214, 244, 234, 168, 80,
88, 175

5.7.5. Operations on Symbols

Operations on symbols have the same semantics as operations on vectors of octets of length T in this specification. Thus, if U and V are two symbols formed by the octets u[0], ..., u[T-1] and v[0], ..., v[T-1], respectively, the sum of symbols U + V is defined to be the component-wise sum of octets, i.e., equal to the symbol D formed by the octets d[0], ..., d[T-1], such that d[i] = u[i] + v[i], 0 <= i < T.

Furthermore, if beta is an octet, the product beta*U is defined to be the symbol D obtained by multiplying each octet of U by beta, i.e.,

d[i] = beta*u[i], 0 <= i < T.

5.7.6. Operations on Matrices

All matrices in this specification have entries that are octets, and thus matrix operations and definitions are defined in terms of the underlying octet arithmetic, e.g., operations on a matrix, matrix rank and matrix inversion.

5.8. Requirements for a Compliant Decoder

If a RaptorQ compliant decoder receives a mathematically sufficient set of encoding symbols generated according to the encoder specification in Section 5.3 for reconstruction of a source block then such a decoder SHOULD recover the entire source block.

A RaptorQ compliant decoder SHALL have the following recovery properties for source blocks with K' source symbols for all values of K' in Table 2 of Section 5.6.

1. If the decoder receives K' encoding symbols generated according to the encoder specification in Section 5.3 with corresponding ESIs chosen independently and uniformly at random from the range of possible ESIs then on average the decoder will fail to recover the entire source block at most 1 out of 100 times.

2. If the decoder receives K'+1 encoding symbols generated according to the encoder specification in Section 5.3 with corresponding ESIs chosen independently and uniformly at random from the range of possible ESIs then on average the decoder will fail to recover the entire source block at most 1 out of 10,000 times.

3. If the decoder receives K'+2 encoding symbols generated according to the encoder specification in Section 5.3 with corresponding ESIs chosen independently and uniformly at random from the range of possible ESIs then on average the decoder will fail to recover the entire source block at most 1 out of 1,000,000 times.

Note that the Example FEC Decoder specified in Section 5.4 fulfills both requirements, i.e.

1. it can reconstruct a source block as long as it receives a mathematically sufficient set of encoding symbols generated

generated according to the encoder specification in Section 5.3;

2. it fulfills the mandatory recovery properties from above.

6. Security Considerations

Data delivery can be subject to denial-of-service attacks by attackers which send corrupted packets that are accepted as legitimate by receivers. This is particularly a concern for multicast delivery because a corrupted packet may be injected into the session close to the root of the multicast tree, in which case the corrupted packet will arrive at many receivers. This is particularly a concern when the code described in this document is used because the use of even one corrupted packet containing encoding data may result in the decoding of an object that is completely corrupted and unusable. It is thus RECOMMENDED that source authentication and integrity checking are applied to decoded objects before delivering objects to an application. For example, a SHA-1 hash [SHA1] of an object may be appended before transmission, and the SHA-1 hash is computed and checked after the object is decoded but before it is delivered to an application. Source authentication SHOULD be provided, for example by including a digital signature verifiable by the receiver computed on top of the hash value. It is also RECOMMENDED that a packet authentication protocol such as TESLA [RFC4082] be used to detect and discard corrupted packets upon arrival. This method may also be used to provide source authentication. Furthermore, it is RECOMMENDED that Reverse Path Forwarding checks be enabled in all network routers and switches along the path from the sender to receivers to limit the possibility of a bad agent successfully injecting a corrupted packet into the multicast tree data path.

Another security concern is that some FEC information may be obtained by receivers out-of-band in a session description, and if the session description is forged or corrupted then the receivers will not use the correct protocol for decoding content from received packets. To avoid these problems, it is RECOMMENDED that measures be taken to prevent receivers from accepting incorrect session descriptions, e.g., by using source authentication to ensure that receivers only accept legitimate session descriptions from authorized senders.

7. IANA Considerations

Values of FEC Encoding IDs and FEC Instance IDs are subject to IANA registration. For general guidelines on IANA considerations as they apply to this document, see [RFC5052]. This document assigns the

Fully-Specified FEC Encoding ID 6 (tbc) under the ietf:rmt:fec: encoding name-space to "RaptorQ Code".

8. Acknowledgements

Thanks are due to Ranganathan (Ranga) Krishnan. Ranga Krishnan has been very supportive in finding and resolving implementation details and in finding the systematic indices. In addition, Habeeb Mohiuddin Mohammed and Antonios Pitarokoilis, both from the Munich University of Technology (TUM) and Alan Shinsato have done two independent implementations of the RaptorQ encoder/decoder that have helped to clarify and to resolve issues with this specification.

9. References

9.1. Normative references

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC4082] Perrig, A., Song, D., Canetti, R., Tygar, J., and B. Briscoe, "Timed Efficient Stream Loss-Tolerant Authentication (TESLA): Multicast Source Authentication Transform Introduction", RFC 4082, June 2005.

[SHA1] "Secure Hash Standard", Federal Information Processing Standards Publication (FIPS PUB) 180-1, April 2005.

[RFC5052] Watson, M., Luby, M., and L. Vicisano, "Forward Error Correction (FEC) Building Block", RFC 5052, August 2007.

9.2. Informative references

[RFC3453] Luby, M., Vicisano, L., Gemmell, J., Rizzo, L., Handley, M., and J. Crowcroft, "The Use of Forward Error Correction (FEC) in Reliable Multicast", RFC 3453, December 2002.

[RFC5053] Luby, M., Shokrollahi, A., Watson, M., and T. Stockhammer, "Raptor Forward Error Correction Scheme for Object Delivery", RFC 5053, October 2007.

Authors' Addresses

Michael Luby
Qualcomm Incorporated
3165 Kifer Road
Santa Clara, CA 95051
U.S.A.

Email: luby@qualcomm.com

Amin Shokrollahi
EPFL
Laboratoire d'algorithmique
EPFL
Station 14
Batiment BC
Lausanne 1015
Switzerland

Email: amin.shokrollahi@epfl.ch

Mark Watson
Qualcomm Incorporated
3165 Kifer Road
Santa Clara, CA 95051
U.S.A.

Email: watson@qualcomm.com

Thomas Stockhammer
Nomor Research
Brecherspitzstrasse 8
Munich 81541
Germany

Email: stockhammer@nomor.de

Lorenz Minder
Qualcomm Incorporated
3165 Kifer Road
Santa Clara, CA 95051
U.S.A.

Email: lminder@qualcomm.com

WHAT IS CLAIMED IS:

1. A method of electronically transmitting data via one or more transmitters capable of outputting an electronic signal, wherein the data to be transmitted is represented by an ordered set of source symbols and the data is transmitted as a sequence of encoded symbols representing at least a portion of the electronic signal, the method comprising:

obtaining, in an electronically readable form, the ordered set of source symbols;

generating a set of intermediate symbols from the ordered set of source symbols, wherein the source symbols can be regenerated from the set of intermediate symbols;

designating sets of the intermediate symbols such that each intermediate symbol is designated as a member of one of the sets of intermediate symbols and there are at least a first set of intermediate symbols and a second set of intermediate symbols, and wherein each set of intermediate symbols has associated with it distinct encoding parameters and has as members at least one intermediate symbol; and

generating a plurality of encoded symbols, wherein an encoded symbol is generated from one or more of the intermediate symbols, wherein at least one encoded symbol is generated, directly or indirectly, from a plurality of intermediate symbols selected from a plurality of the sets of intermediate symbols.

2. The method of claim 1, wherein the first set of intermediate symbols are designated as symbols for belief propagation decoding and the second set of intermediate symbols are designated as symbols to be inactivated for belief propagation decoding.

3. The method of claim 1, wherein each encoded symbol is generated from a combination of a first symbol generated from one or more of the first set of intermediate symbols and a second symbol generated from one or more of the second set of intermediate symbols.

4. The method of claim 3, wherein the distinct encoding parameters comprise at least distinct degree distributions, such that each encoded symbol is generated from a combination of a first symbol generated from one or more of the first set of intermediate symbols having a first degree distribution and a second symbol generated from one or more of the second set of intermediate symbols having a second degree distribution different from the first degree distribution.

5. The method of claim 3, wherein the first symbol is generated using a chain reaction encoding process applied to the first set of intermediate symbols.

6. The method of claim 3, wherein the second symbol is an XOR of a fixed number of symbols chosen randomly from the second set of intermediate symbols.

7. The method of claim 3, wherein the second symbol is an XOR of a first number of symbols chosen randomly from the second set of intermediate symbols, and wherein the first number depends on a second number equal to a number of the symbols chosen from the first set to generate the first symbol.

8. The method of claim 3, wherein the combination is the XOR of the first symbol and the second symbol.

9. The method of claim 1, wherein the intermediate symbols comprise the ordered set of source symbols and a set of redundant source symbols generated from the ordered set of source symbols.

10. The method of claim 9, wherein at least some of the redundant symbols are generated using a GF[2] operations and other redundant symbols are generated using GF[256] operations.

11. The method of claim 1, wherein the intermediate symbols are generated, during encoding, from the source symbols using a decoding process, wherein the decoding process is based on a linear set of relations between the intermediate symbols and the source symbols.

12. The method of claim 11, wherein at least some of the linear relations are relations over GF[2] and other linear relations are relations over GF[256].

13. The method of claim 1, wherein the number of distinct encoded symbols that can be generated from a given ordered set of source symbols is independent of the number of source symbols in that ordered set.

14. The method of claim 1, wherein an average number of symbol operations performed to generate an encoded symbol is bounded by a constant independent of the number of source symbols in that ordered set.

15. The method of claim 1, wherein the first set of symbols is more than an order of magnitude larger than the second set of symbols.

16. A method of receiving data from a source, wherein the data is received at a destination over a packet communication channel, and wherein the data representable by a set of encoded symbols derived from an ordered set of source symbols representing the data sent from the source to the destination, the method comprising:

obtaining the set of received encoded symbols;

decoding a set of intermediate symbols from the set of received encoded symbols;

associating each of the intermediate symbols with a set of intermediate symbols, wherein the intermediate symbols are associated into at least two sets, and wherein one set is designated as a set of permanently inactive symbols for purposes of scheduling a decoding process to recover the intermediate symbols from the received encoded symbols; and

recovering at least some of the source symbols of the ordered set of source symbols from the set of intermediate symbols according to the decoding process.

17. The method of claim 16, wherein the decoding process comprises at least a first decoding phase, wherein a set of reduced encoded symbols are generated that depend on a second set of permanently inactive symbols and a third set of dynamically inactive symbols that is a subset of the first set of symbols, and a second decoding phase, wherein the set of reduced encoded symbols is used to decode the second set of permanently inactive symbols and the third set of dynamically inactive symbols, and a third decoding phase, wherein the decoded second set of permanently inactive symbols and the third set of dynamically inactive symbols and the set of received encoded symbols is used to decode at least some of the remaining intermediate symbols that are in the first set of symbols.

18. The method of claim 17, wherein the first decoding phase uses belief propagation decoding combined with inactivation decoding, and/or the second decoding phase uses Gaussian elimination.

19. The method of claim 17, wherein the third decoding phase uses back substitution or a back sweep followed by a forward sweep.

20. The method of claim 17, wherein the decoding process operates on the third set of dynamically inactive symbols considering that the number of symbols in third set of dynamically inactive symbols is less than 10% of the number of source symbols and/or less than 10% of the number of symbols in the second set of permanently inactive symbols.

21. The method of claim 16, wherein the received encoded symbols are operated on as LDPC code generated symbols or Reed-Solomon code generated symbols.

22. The method of claim 16, wherein each received encoded symbol of the set of received encoded symbols is operated on as being a combination of a first symbol generated from one or more of the first set of symbols and a second symbol generated from one or more of the second set of symbols.

23. The method of claim 22, wherein each received encoded symbol is operated on as the combination being an XOR of the first symbol and the second symbol.

24. The method of claim 22, wherein each received encoded symbol is operated on as the second symbol being an XOR of a fixed number of symbols that was chosen randomly from the second set.

25. The method of claim 22, wherein each received encoded symbol is operated on as the second symbol being an XOR of a first number of symbols that was chosen randomly from the second set, wherein the first number of symbols depends on the second number of symbols that was chosen from the first set to generate the first symbol.

26. The method of claim 22, wherein the decoding process operates as if the first symbol was chosen based on a chain reaction code from the first set of symbols.

27. The method of claim 16, wherein the decoding process operates as if the size of the second set of permanently inactive symbols is proportional to the square root of the number of source symbols.

28. The method of claim 16, wherein the decoding process operates as if the intermediate symbols comprise the ordered set of source symbols and a set of redundant symbols generated from the ordered set of source symbols.

29. The method of claim 28, wherein the decoding process operates as if at least some of the redundant symbols were generated using GF[2] operations and other redundant symbols were generated using GF[256] operations.

30. The method of claim 16, wherein the decoding process operates as if the intermediate symbols comprise the ordered set of source symbols.

31. The method of claim 16, wherein the decoding process operates as if the intermediate symbols are symbols that were generated from the source symbols using a decoding process based on a linear set of relations between the intermediate symbols and the source symbols.

32. The method of claim 31, wherein the decoding process operates as if at least some of the linear relations are relations over GF[2] and other linear relations are relations over GF[256].

33. The method of claim 16, wherein the decoding process operates as if the number of different possible encoded symbols that can be received is independent of the number of source symbols in the ordered set.

34. The method of claim 16, wherein an average number of symbol operations performed to decode the set of source symbols from the set of received encoded symbols is bounded by a constant times the number of source symbols, wherein the constant is independent of the number of source symbols.

35. The method of claim 16, wherein the decoding process operates as if the number of symbols in the first set of symbols is more than an order of magnitude larger than the number of symbols in the second set of permanently inactive symbols.

36. The method of claim 16, wherein the decoding process operates such that recovery of all of the set of K source symbols from a set of N=K+A encoded symbols, for some K, N and A, has a probability of success of at least a lower bound of 1-(0.01)^(A+1) for A=0, 1 or 2, with the lower bound being independent of the number of source symbols.

37. A method for serving a file using a server coupled to a data network, wherein serving includes organizing data of the file into one or more blocks, generating one or more

encoded symbols for a block based on the data of the block, and wherein at least one block is physically or logically organized into a plurality of sub-blocks and at least one encoded symbol is physically or logically organized into a plurality of sub-symbols, the method comprising:

partitioning an input file into an integer number of blocks, wherein each block includes at least one sub-block, and wherein each sub-block includes at least one source symbol;

determining a value, WS, representing a maximum size for a sub-block based on a memory constraint;

determining a value SS, wherein SS*AL represents a lower bound for sub-symbol size, in units of a preferred memory unit size, AL;

determining which blocks of the integer number of blocks is to be organized into a plurality of sub-blocks, and for each such block, organizing the block into a plurality of sub-blocks having a size determined by the available space within packets for encoded symbols that are to be sent, a symbol size that is to be used within each sent packet, in a manner to ensure that a number of source symbols for source blocks is equal within a threshold and the number is equal to the number, Kt, of source symbols in the file and to ensure that the sub-symbol size of each sub-block is at most SS*AL and to ensure that the size of each sub-block is at most WS;

generating encoded symbols from blocks, wherein sub-symbols are generated from sub-blocks such that each encoded symbol depends on data from one block; and

outputting the generated encoded symbols.

38. A method for recovering a block of data at a receiver using a client coupled to a data network, wherein a block includes a grouping of one or more sub-blocks, the method comprising:

receiving a plurality of encoded symbols generated from the block, wherein each encoded symbol includes a plurality of sub-symbols generated from at least one sub-block using a common set of operations;

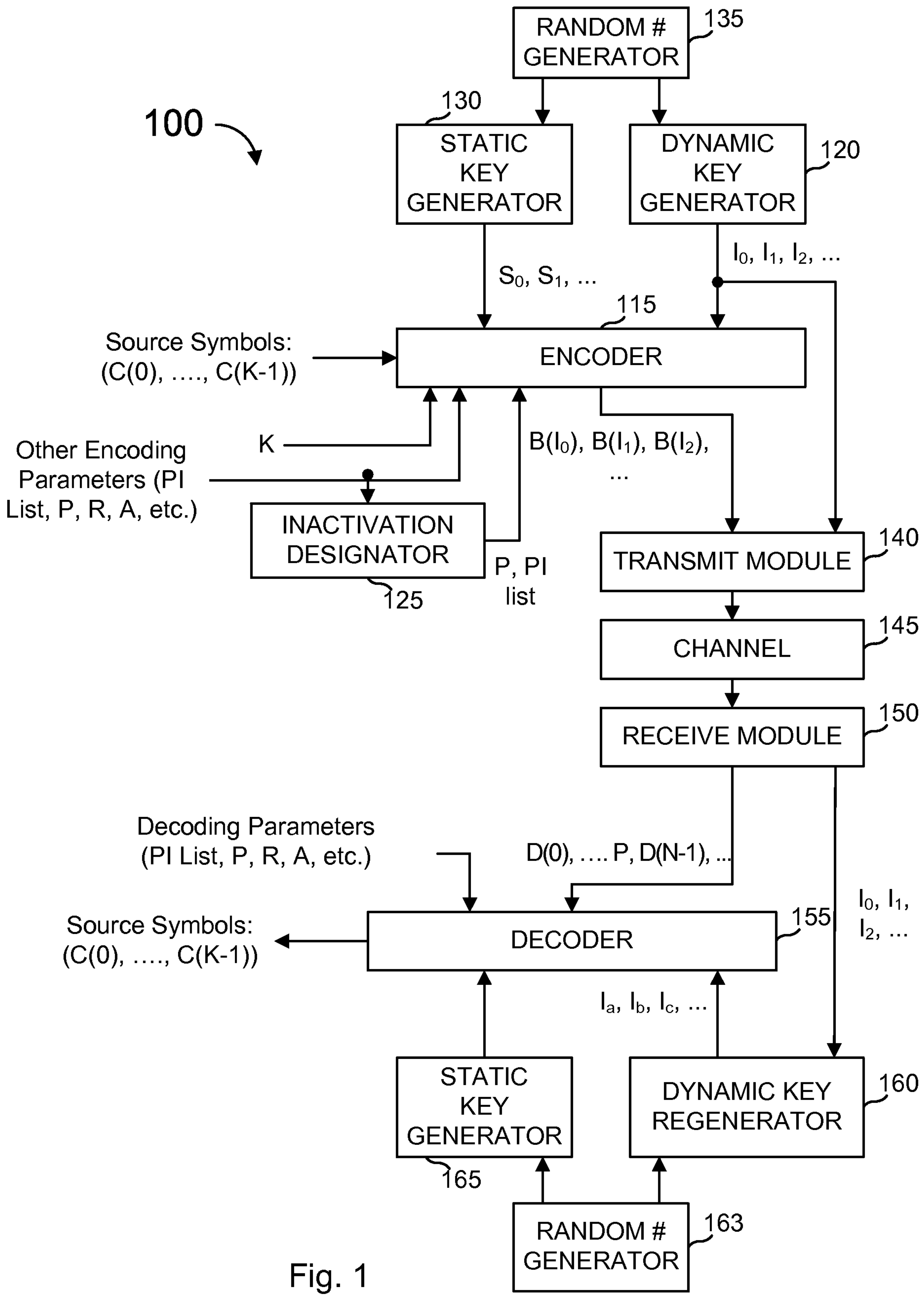
determining a value, WS, representing a maximum size for a sub-block based on a memory constraint;

determining a value SS, wherein SS*AL represents a lower bound for sub-symbol size, in units of a preferred memory unit size, AL;

determining which blocks of the integer number of blocks organized into a plurality of sub-blocks, and for each such block, organizing the block into a plurality of sub-blocks having a size determined by a first parameter set by a sender representing available space within packets, a second parameter representing a symbol size used within each packet, the parameters being such that a number of source symbols for source blocks is equal within a threshold and the number is equal to the number, Kt, of source symbols in the file;

decoding blocks from received encoded symbols, wherein sub-blocks are decoded from sub-symbols and the sub-blocks form blocks, wherein the sub-symbol size of each sub-block is at most SS*AL and the size of each sub-block is at most WS; and

outputting the decoded blocks.



100
RANDOM # GENERATOR
135
130
STATIC KEY GENERATOR
DYNAMIC KEY GENERATOR
120
I_0, I_1, I_2, ...
S_0, S_1, ...
115
Source Symbols: (C(0), ..., C(K-1))
ENCODER
K
Other Encoding Parameters (PI List, P, R, A, etc.)
$B(I_0)$, $B(I_1)$, $B(I_2)$, ...
INACTIVATION DESIGNATOR
125
P, PI list
TRANSMIT MODULE
140
CHANNEL
145
RECEIVE MODULE
150
Decoding Parameters (PI List, P, R, A, etc.)
D(0), ... P, D(N-1), ...
I_0, I_1, I_2, ...
Source Symbols: (C(0), ..., C(K-1))
DECODER
155
I_a, I_b, I_c, ...
STATIC KEY GENERATOR
165
DYNAMIC KEY REGENERATOR
160
RANDOM # GENERATOR
163

Fig. 1

DATA VALUES USED

K	Number of source symbols
R	Number of redundant symbols
L	Number of intermediate symbols (L = K+R)
S	Number of LDPC symbols
H	Number of HDPC symbols
P	Number of permanently inactive (PI) symbols
N	Number of received encoded symbols
A	Number of overhead symbols (A = N-K)

SYMBOL ARRAYS

(C(0), …, C(K-1))	Source Symbols
(C(K), …, C(L-1))	Redundant Symbols
(C(0), …, C(L-1))	Intermediate Symbols
(C(0), …, C(L-P-1))	LT Intermediate Symbols
(C(L-P), …, C(L-1))	Permanently Inactive (PI) Symbols (i.e., "PI List")
(D(0), …, D(N-1))	Encoded Symbols Received by Decoder 155

Fig. 2

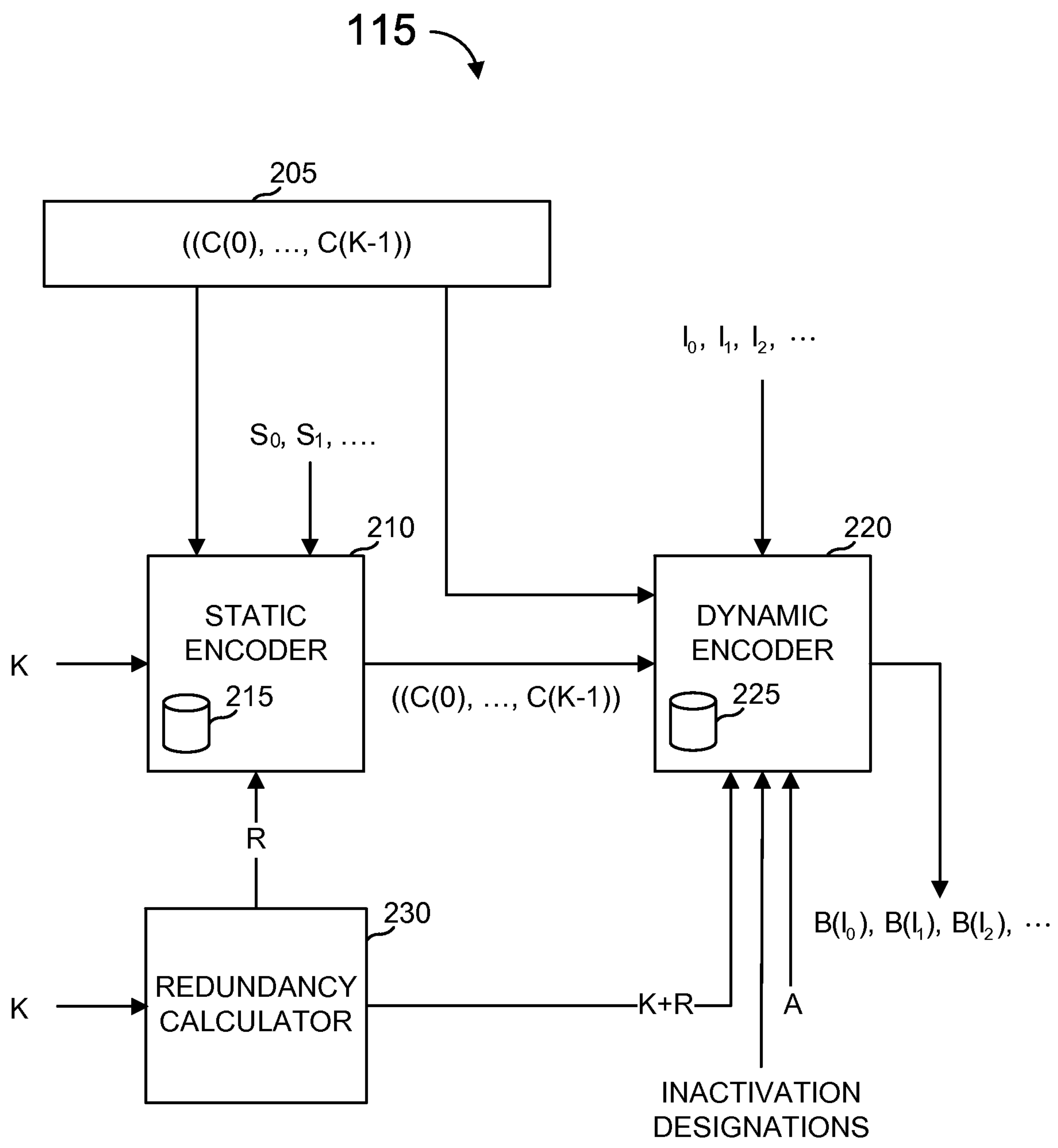

115
205
((C(0), …, C(K-1))
I_0, I_1, I_2, …
S_0, S_1, ….
210
220
K
STATIC ENCODER
215
((C(0), …, C(K-1))
DYNAMIC ENCODER
225
R
230
B(I_0), B(I_1), B(I_2), …
K
REDUNDANCY CALCULATOR
K+R
A
INACTIVATION DESIGNATIONS

Fig. 3

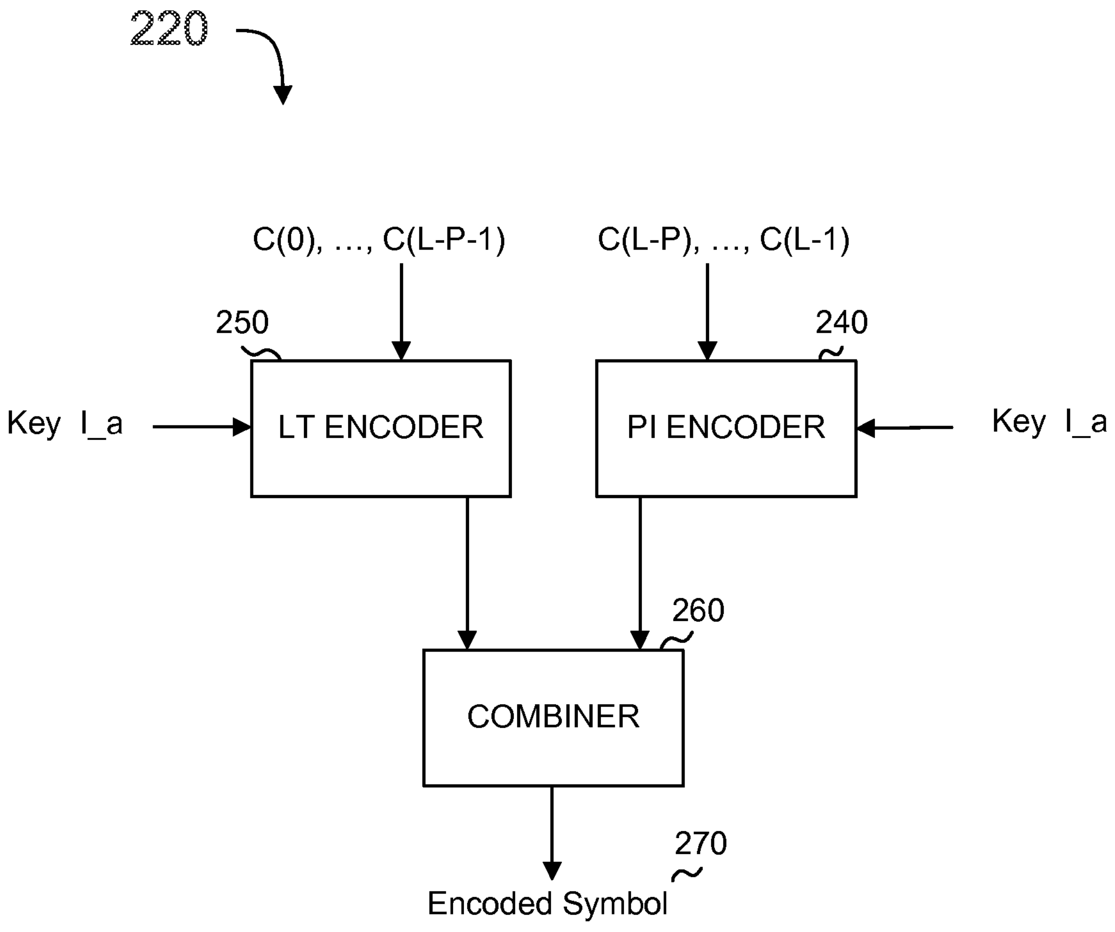

220
C(0), …, C(L-P-1)
C(L-P), …, C(L-1)
250
240
Key I_a
LT ENCODER
PI ENCODER
Key I_a
260
COMBINER
270
Encoded Symbol

Fig. 4

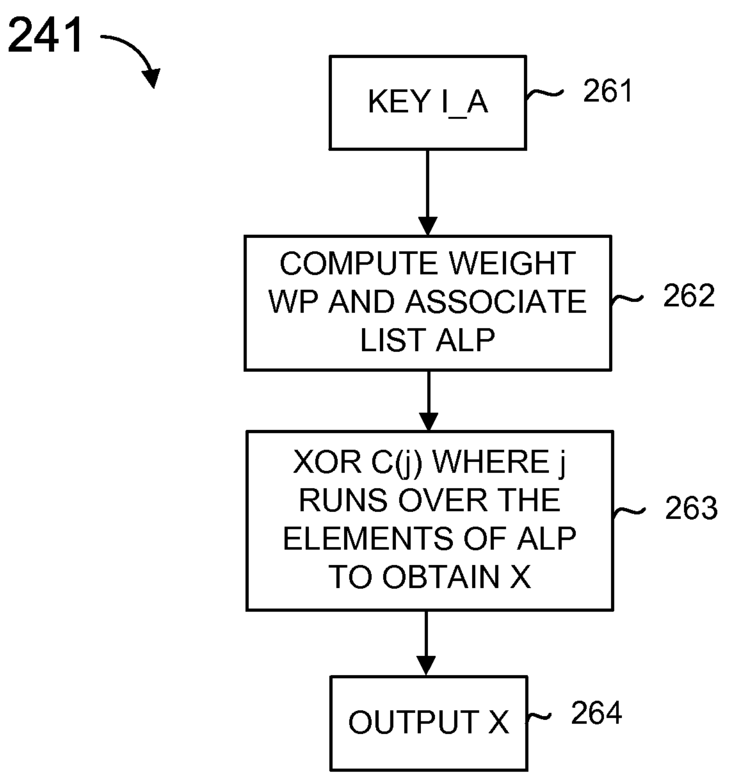

241
KEY I_A
261
COMPUTE WEIGHT WP AND ASSOCIATE LIST ALP
262
XOR C(j) WHERE j RUNS OVER THE ELEMENTS OF ALP TO OBTAIN X
263
OUTPUT X
264

Fig. 5

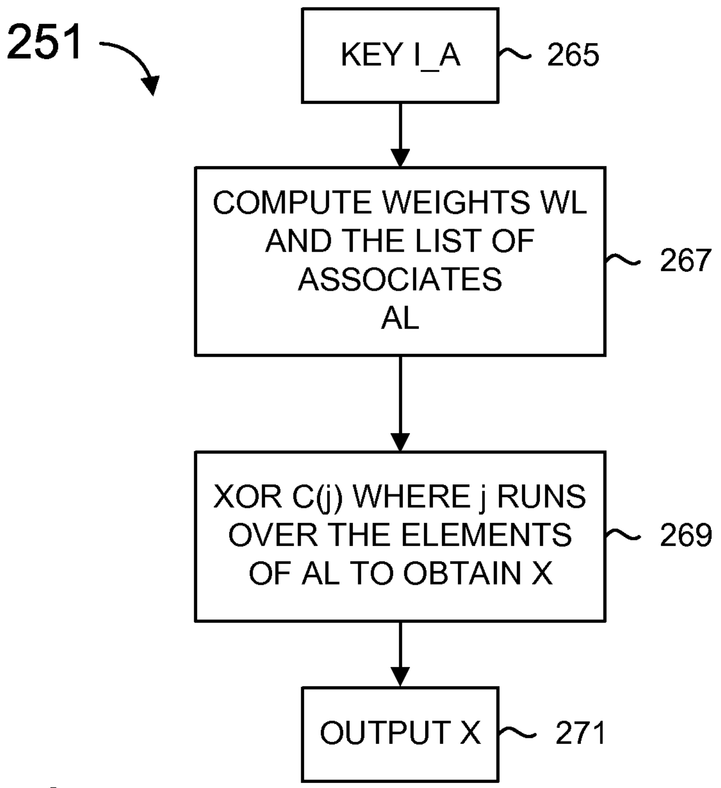

251
KEY I_A
265
COMPUTE WEIGHTS WL AND THE LIST OF ASSOCIATES AL
267
XOR C(j) WHERE j RUNS OVER THE ELEMENTS OF AL TO OBTAIN X
269
OUTPUT X
271

Fig. 6

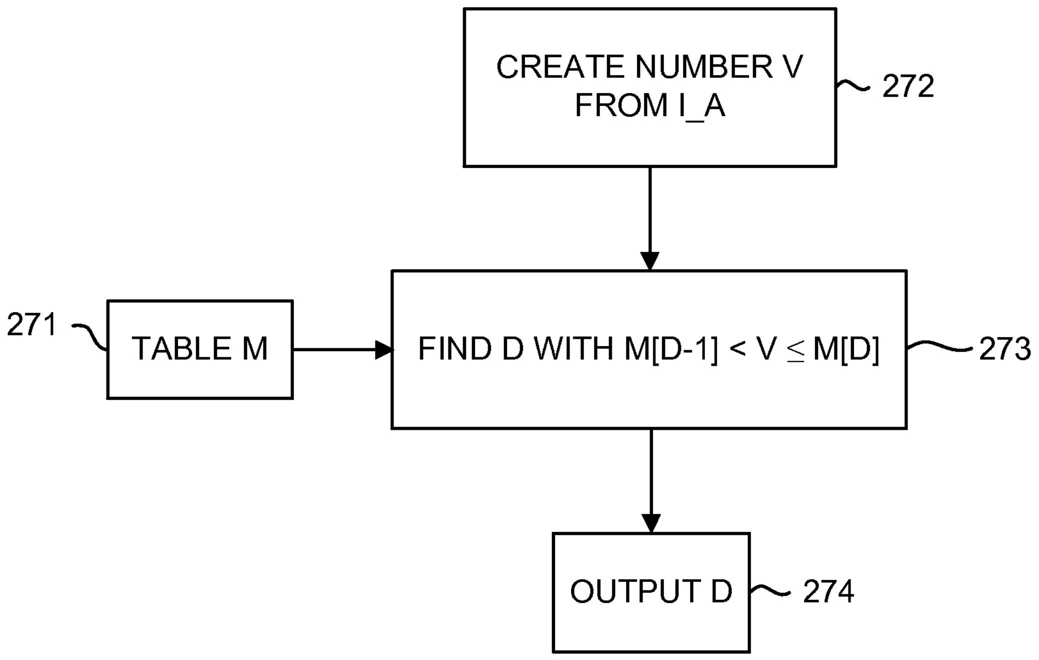

CREATE NUMBER V FROM I_A
272
271
TABLE M
FIND D WITH M[D-1] < V ≤ M[D]
273
OUTPUT D
274

Fig. 7

d	*m[d]*
30	1048576
29	1017662
28	1016370
27	1014983
26	1013490
25	1011876
24	1010129
23	1008229
22	1006157
21	1003887
20	1001391
19	998631
18	995565
17	992138
16	988283
15	983914
14	978921
13	973160
12	966438
11	958494
10	948962
9	937311
8	922747
7	904023
6	879057
5	844104
4	791675
3	704294
2	529531
1	5243
0	0

Fig. 8

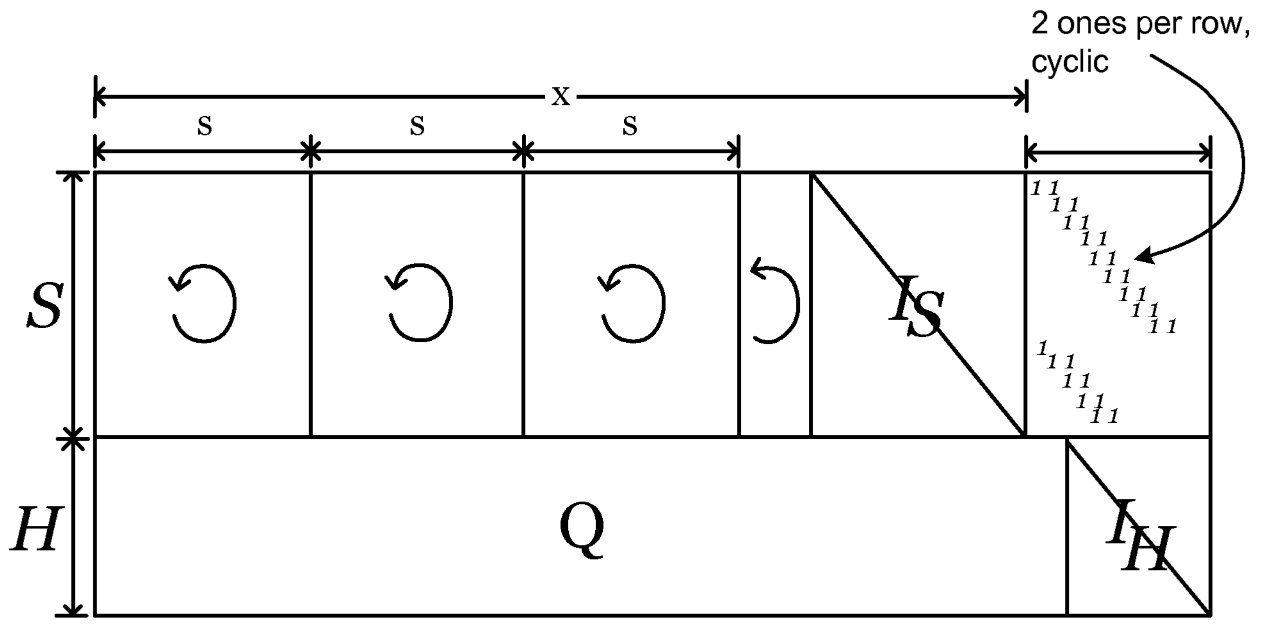


Fig. 9

$$Q = (\Delta_1 | \Delta_2 | \cdots | \Delta_{k+S-1} | Y) \cdot \Gamma, \Gamma = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ \alpha & 1 & 0 & \cdots & 0 & 0 \\ 0 & \alpha & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & \alpha & 1 \end{pmatrix}^{-1}, \ Y = \begin{pmatrix} a^0 \\ a^1 \\ \vdots \\ \alpha^{H-2} \\ a^{H-1} \end{pmatrix}$$

Fig. 10

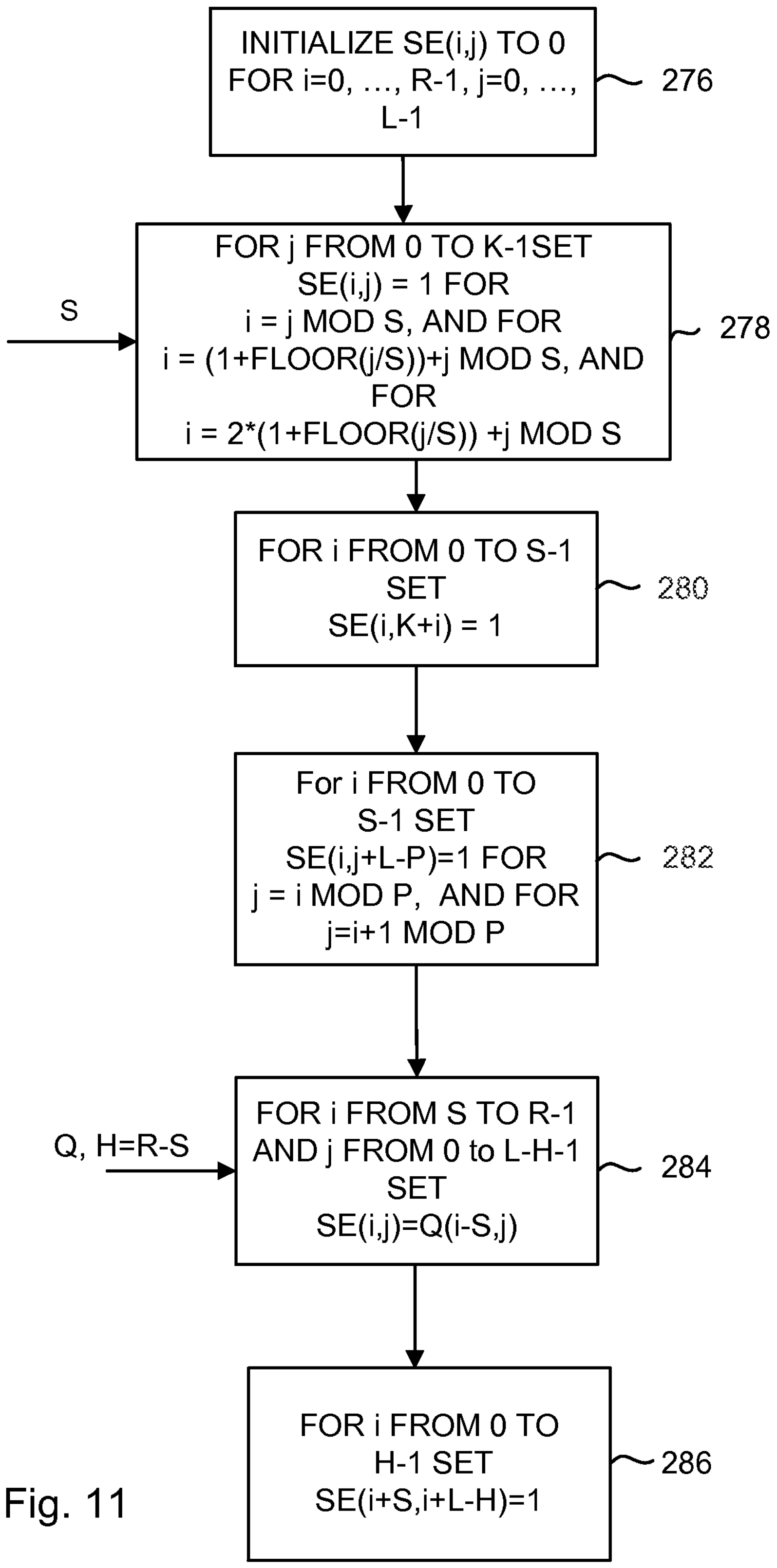

INITIALIZE SE(i,j) TO 0 FOR i=0, …, R-1, j=0, …, L-1
276
S
FOR j FROM 0 TO K-1SET SE(i,j) = 1 FOR i = j MOD S, AND FOR i = (1+FLOOR(j/S))+j MOD S, AND FOR i = 2*(1+FLOOR(j/S)) +j MOD S
278
FOR i FROM 0 TO S-1 SET SE(i,K+i) = 1
280
For i FROM 0 TO S-1 SET SE(i,j+L-P)=1 FOR j = i MOD P, AND FOR j=i+1 MOD P
282
Q, H=R-S
FOR i FROM S TO R-1 AND j FROM 0 to L-H-1 SET SE(i,j)=Q(i-S,j)
284
FOR i FROM 0 TO H-1 SET SE(i+S,i+L-H)=1
286

Fig. 11

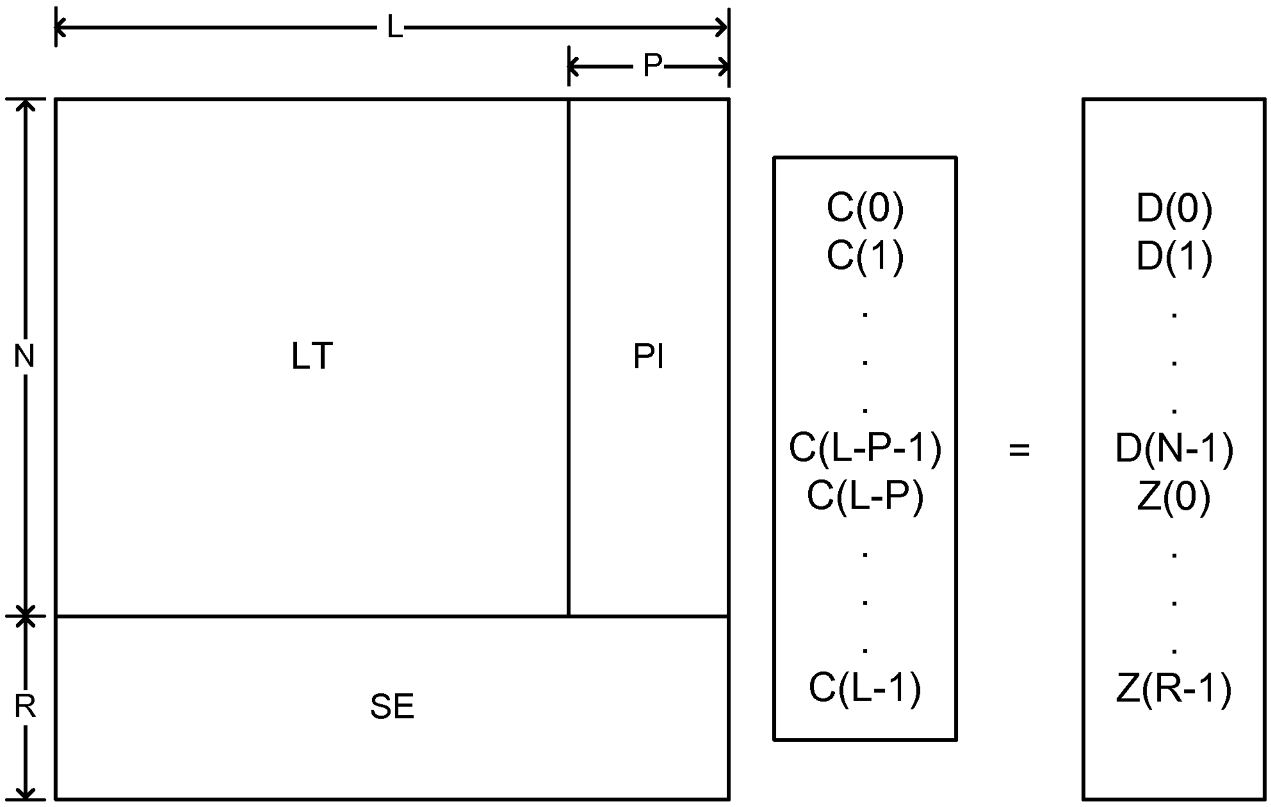


Fig. 12

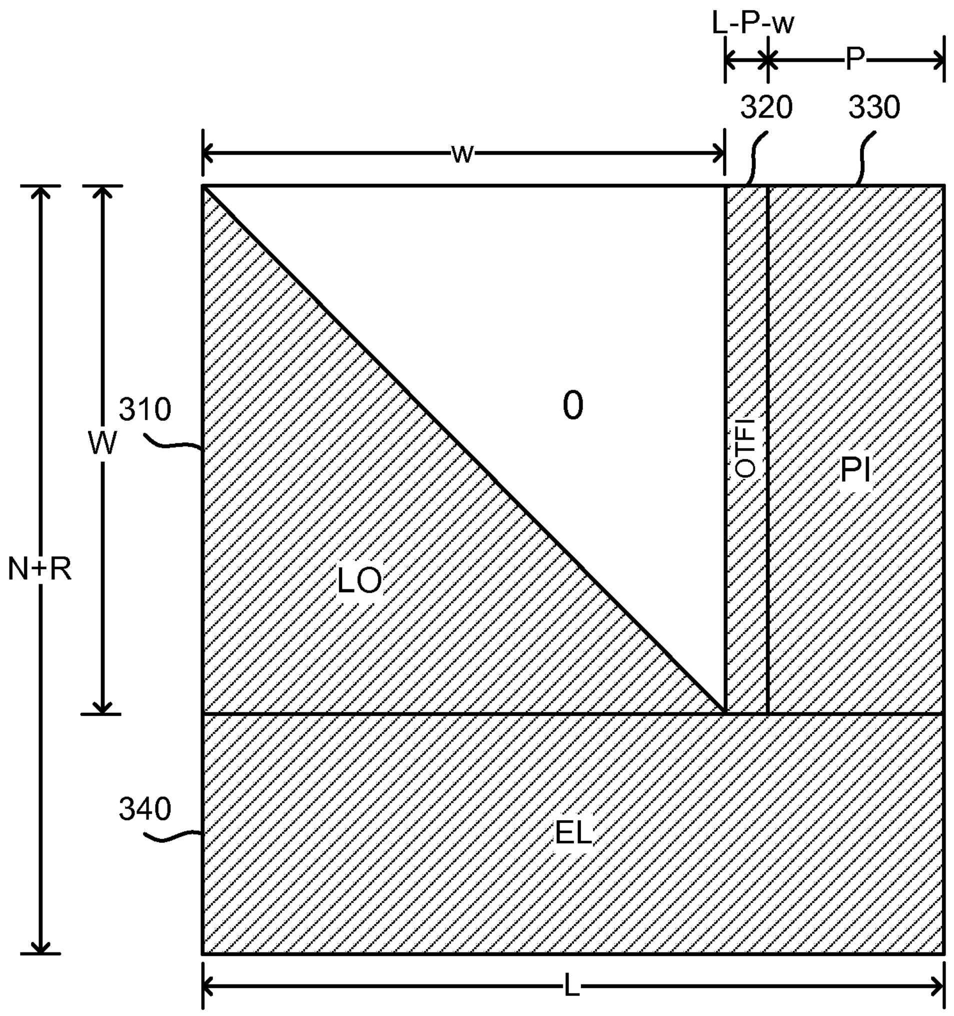


Fig. 13

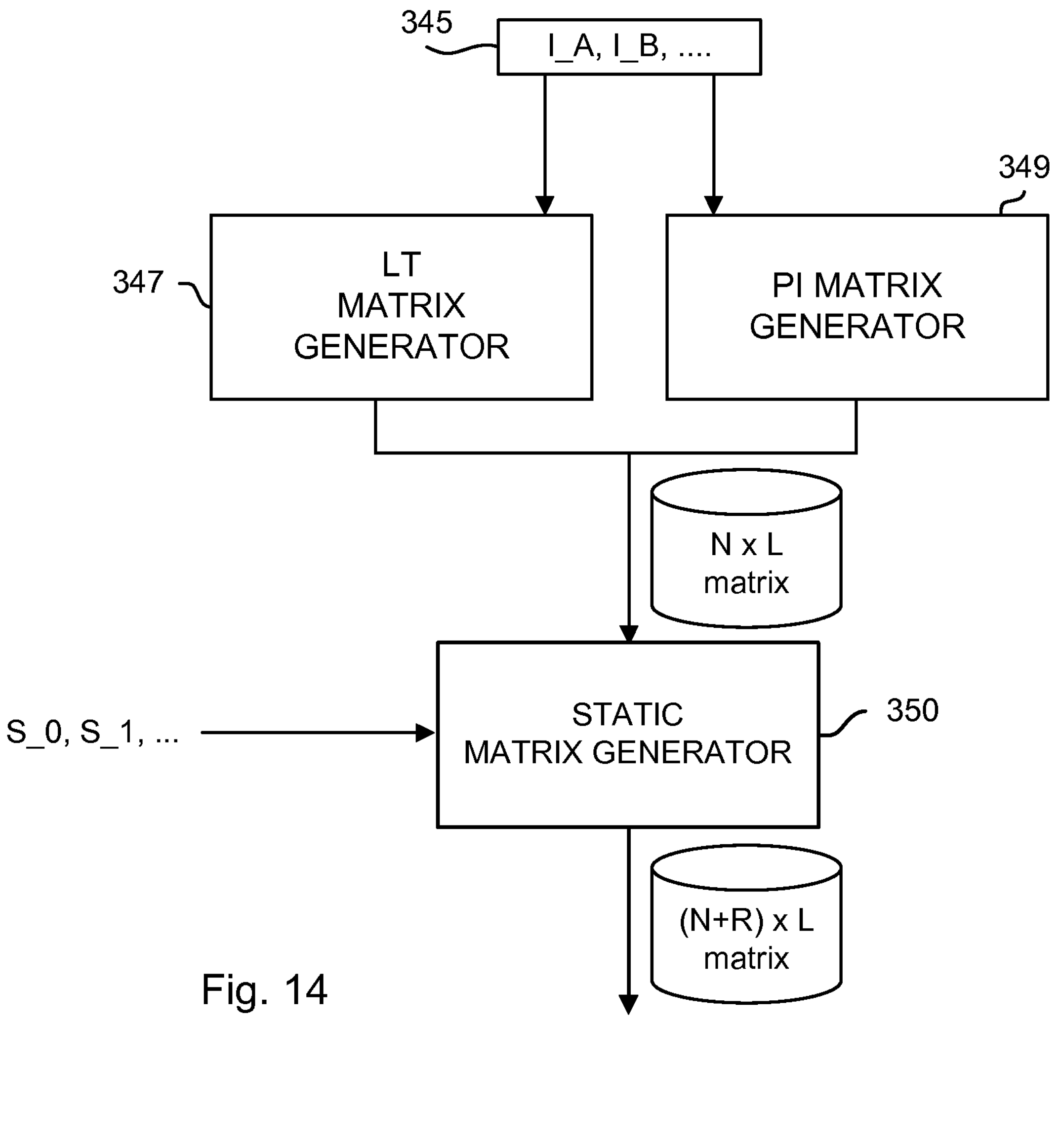

345
I_A, I_B, ...
349
347
LT MATRIX GENERATOR
PI MATRIX GENERATOR
N x L matrix
S_0, S_1, ...
STATIC MATRIX GENERATOR
350
(N+R) x L matrix

Fig. 14

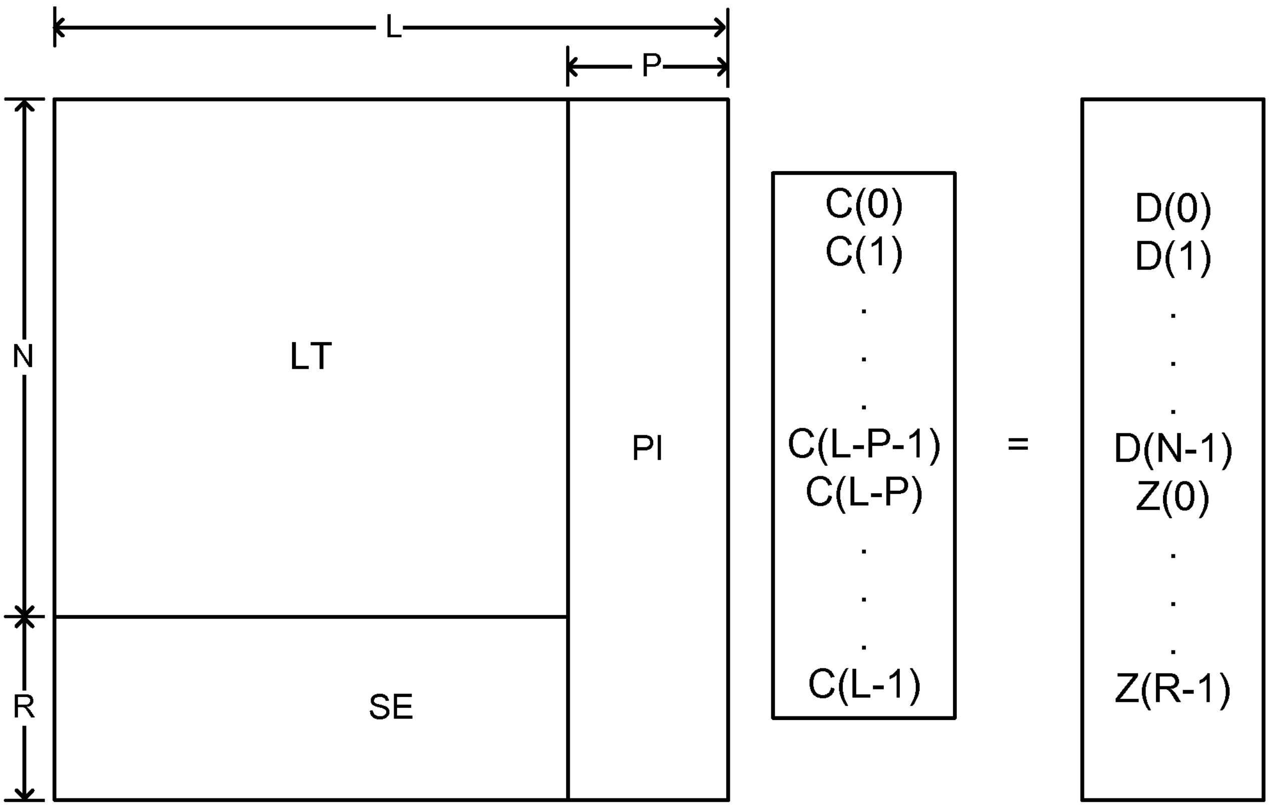


Fig. 15

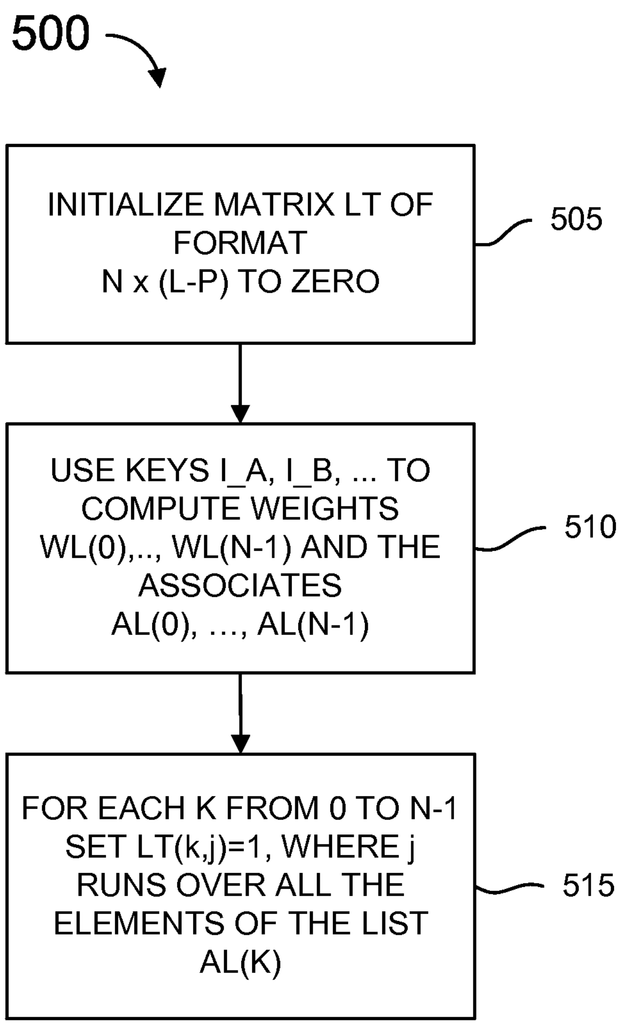

500
INITIALIZE MATRIX LT OF FORMAT N x (L-P) TO ZERO
505
USE KEYS I_A, I_B, ... TO COMPUTE WEIGHTS WL(0),.., WL(N-1) AND THE ASSOCIATES AL(0), …, AL(N-1)
510
FOR EACH K FROM 0 TO N-1 SET LT(k,j)=1, WHERE j RUNS OVER ALL THE ELEMENTS OF THE LIST AL(K)
515

Fig. 16

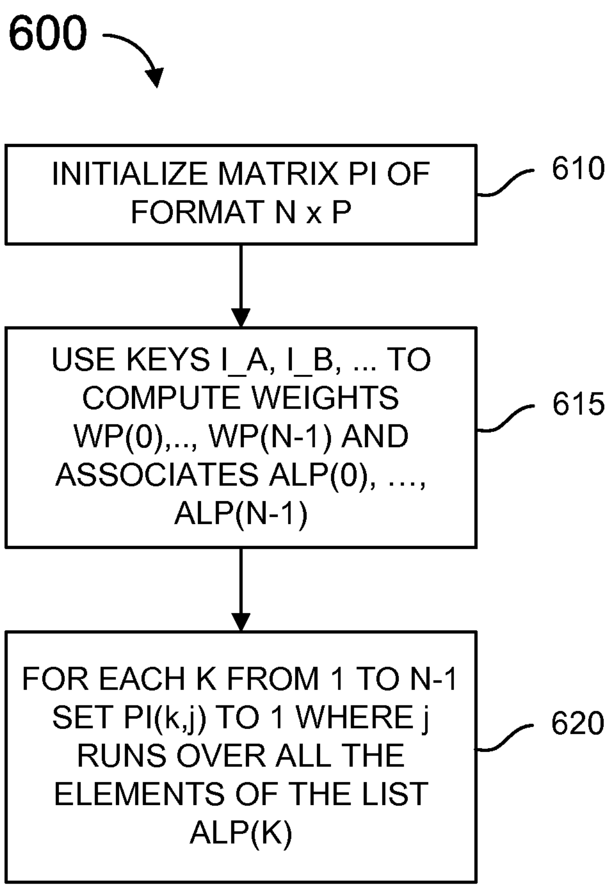

600
INITIALIZE MATRIX PI OF FORMAT N x P
610
USE KEYS I_A, I_B, ... TO COMPUTE WEIGHTS WP(0),.., WP(N-1) AND ASSOCIATES ALP(0), …, ALP(N-1)
615
FOR EACH K FROM 1 TO N-1 SET PI(k,j) TO 1 WHERE j RUNS OVER ALL THE ELEMENTS OF THE LIST ALP(K)
620

Fig. 17

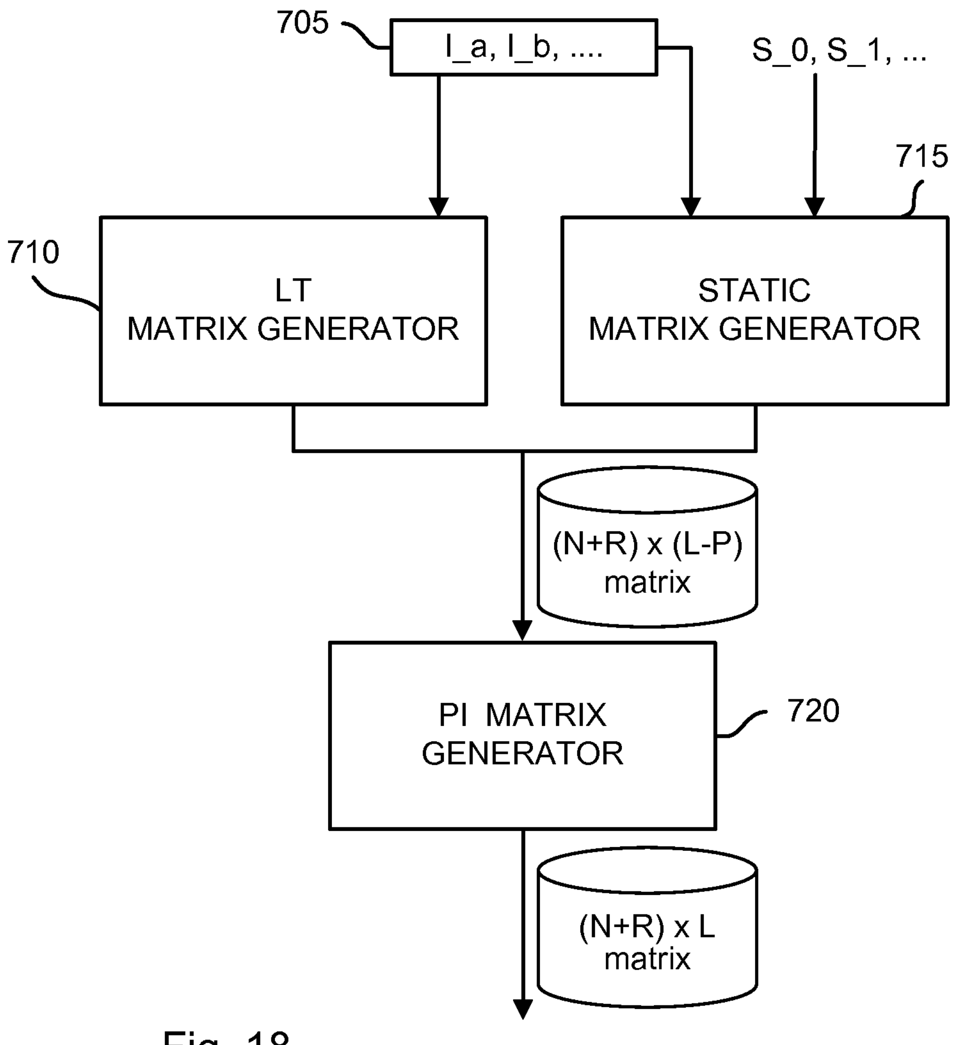

705
I_a, I_b, ...
S_0, S_1, ...
715
710
LT
MATRIX GENERATOR
STATIC
MATRIX GENERATOR
(N+R) x (L-P)
matrix
PI MATRIX
GENERATOR
720
(N+R) x L
matrix

Fig. 18

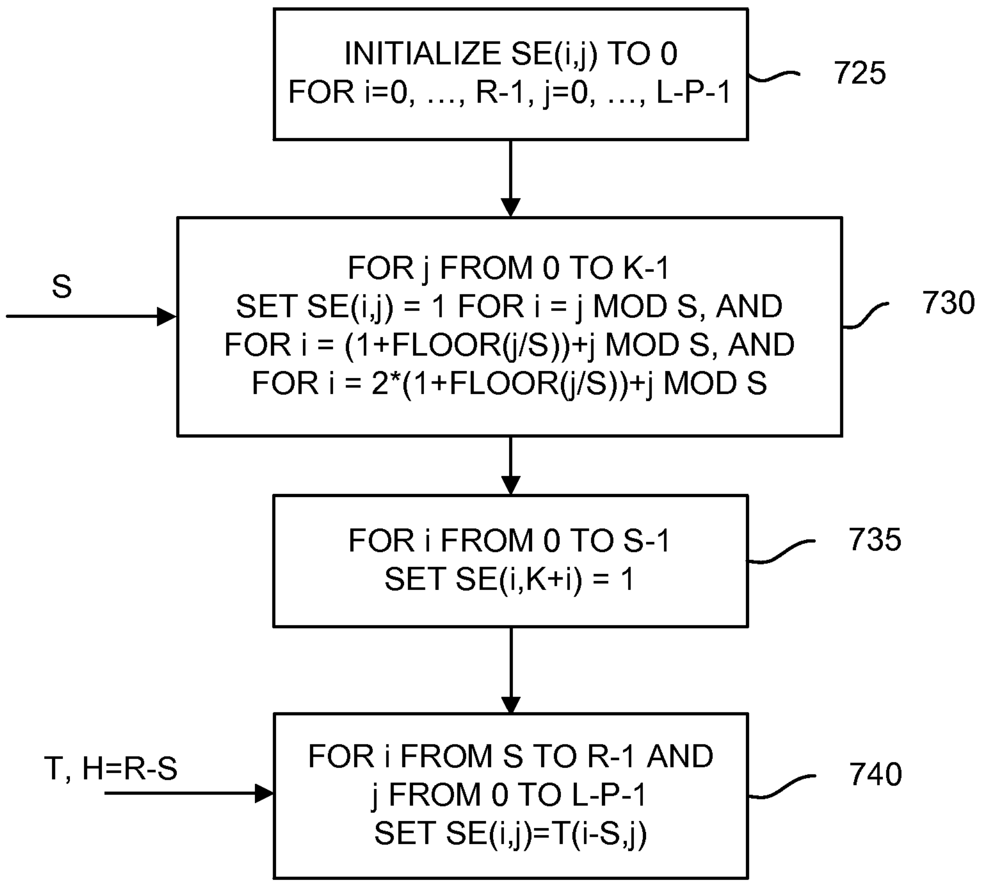

INITIALIZE SE(i,j) TO 0
FOR i=0, …, R-1, j=0, …, L-P-1
725
S
FOR j FROM 0 TO K-1
SET SE(i,j) = 1 FOR i = j MOD S, AND
FOR i = (1+FLOOR(j/S))+j MOD S, AND
FOR i = 2*(1+FLOOR(j/S))+j MOD S
730
FOR i FROM 0 TO S-1
SET SE(i,K+i) = 1
735
T, H=R-S
FOR i FROM S TO R-1 AND
j FROM 0 TO L-P-1
SET SE(i,j)=T(i-S,j)
740

Fig. 19

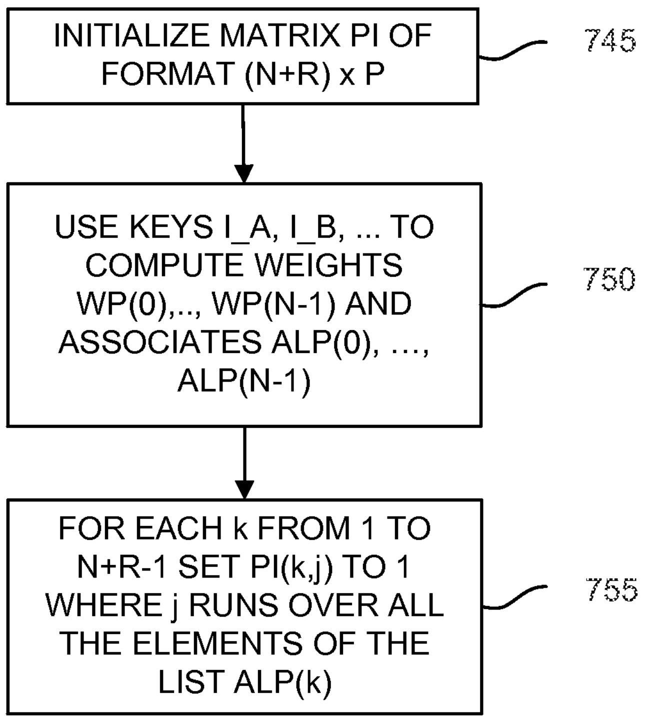

INITIALIZE MATRIX PI OF
FORMAT (N+R) x P
745
USE KEYS I_A, I_B, ... TO
COMPUTE WEIGHTS
WP(0),.., WP(N-1) AND
ASSOCIATES ALP(0), …,
ALP(N-1)
750
FOR EACH k FROM 1 TO
N+R-1 SET PI(k,j) TO 1
WHERE j RUNS OVER ALL
THE ELEMENTS OF THE
LIST ALP(k)
755

Fig. 20

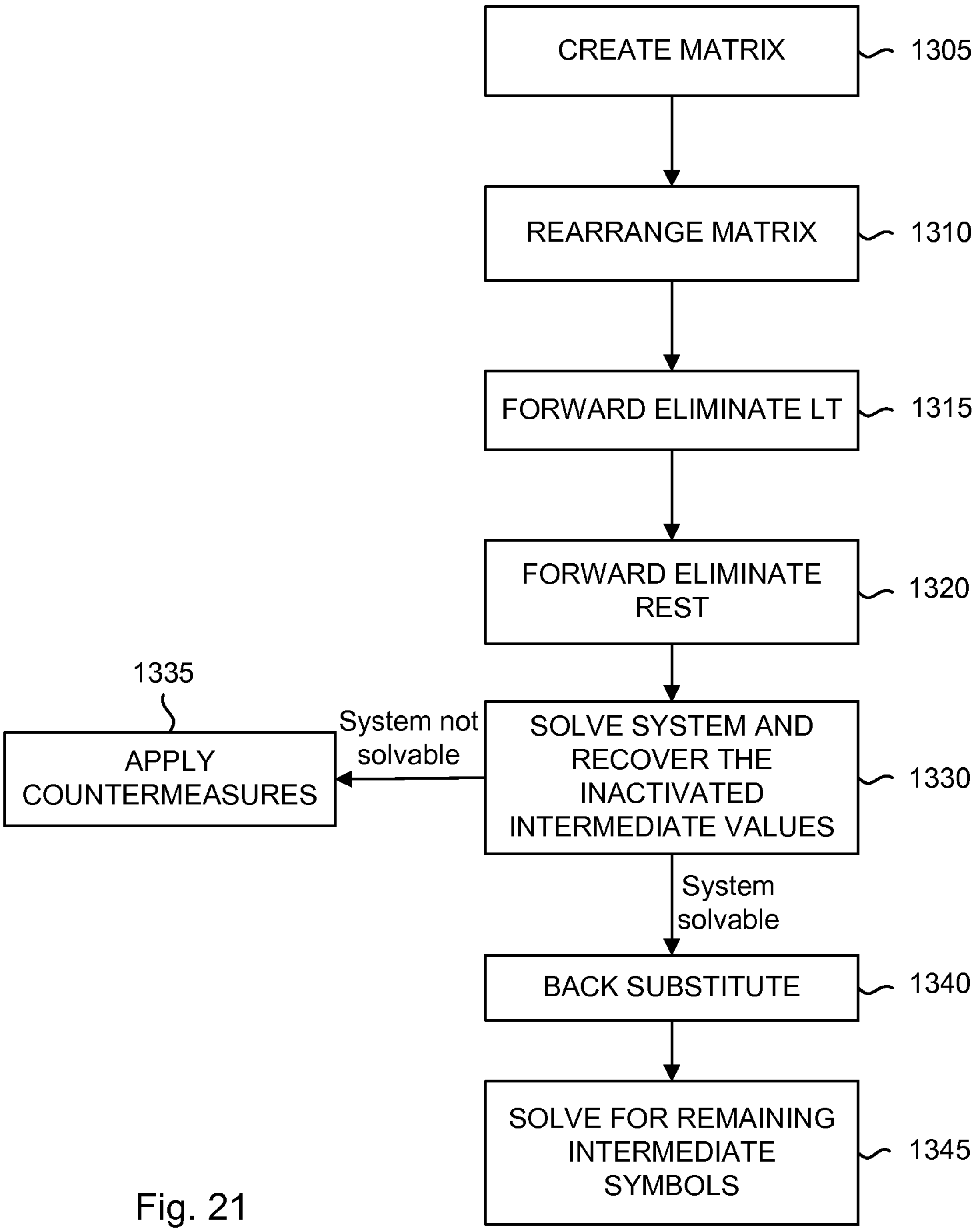

CREATE MATRIX
1305
REARRANGE MATRIX
1310
FORWARD ELIMINATE LT
1315
FORWARD ELIMINATE REST
1320
SOLVE SYSTEM AND RECOVER THE INACTIVATED INTERMEDIATE VALUES
1330
System not solvable
APPLY COUNTERMEASURES
1335
System solvable
BACK SUBSTITUTE
1340
SOLVE FOR REMAINING INTERMEDIATE SYMBOLS
1345

Fig. 21

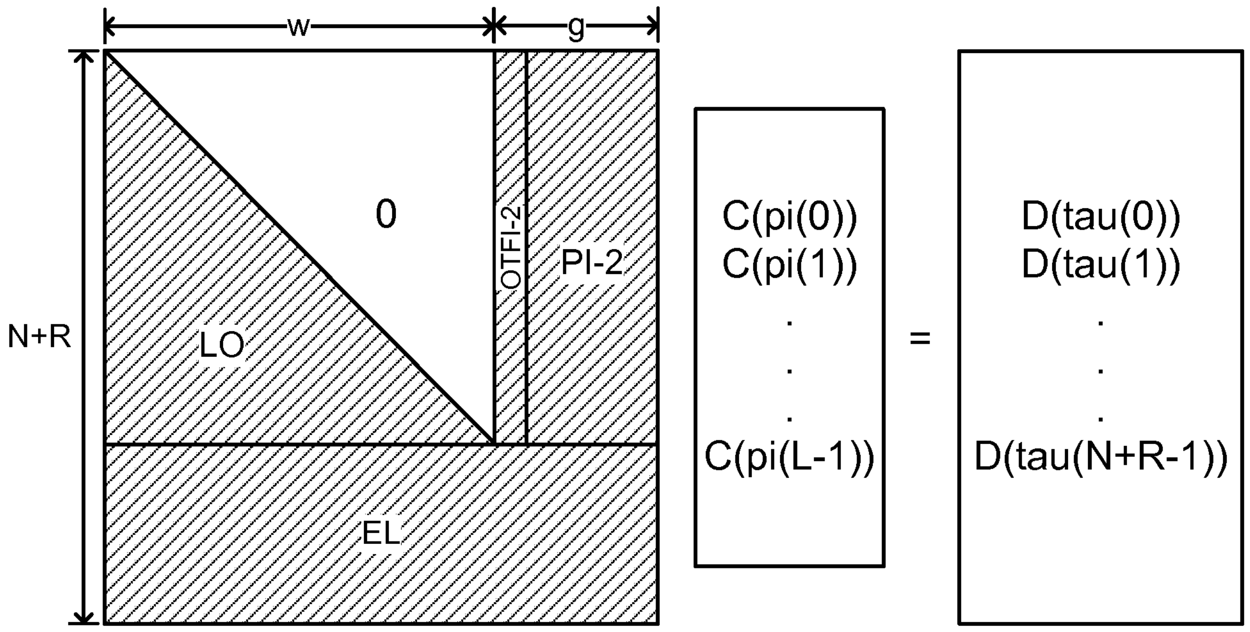


Fig. 22

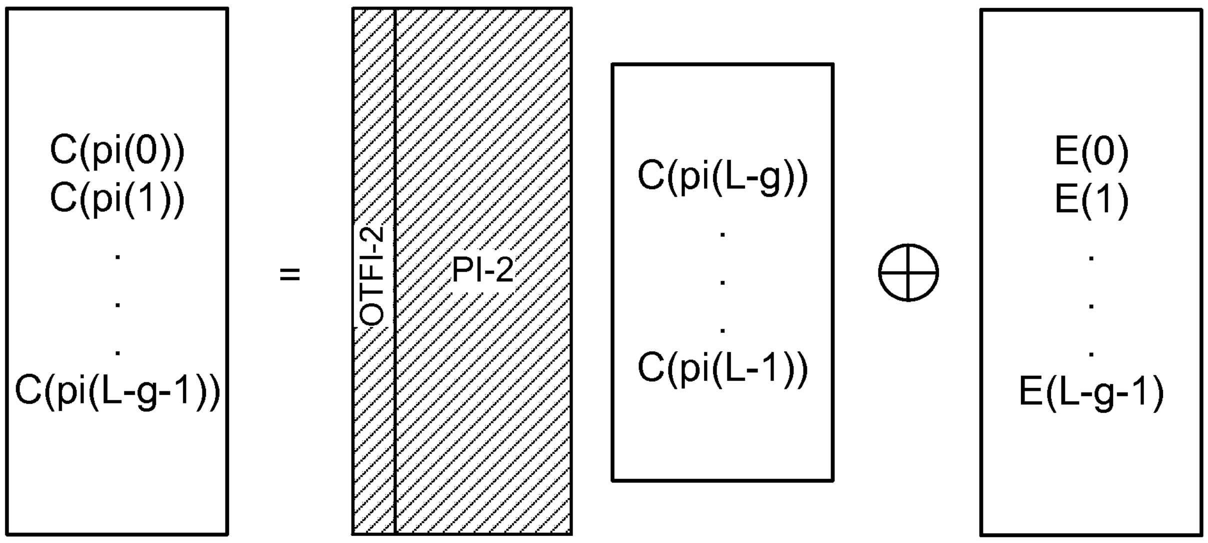


Fig. 23

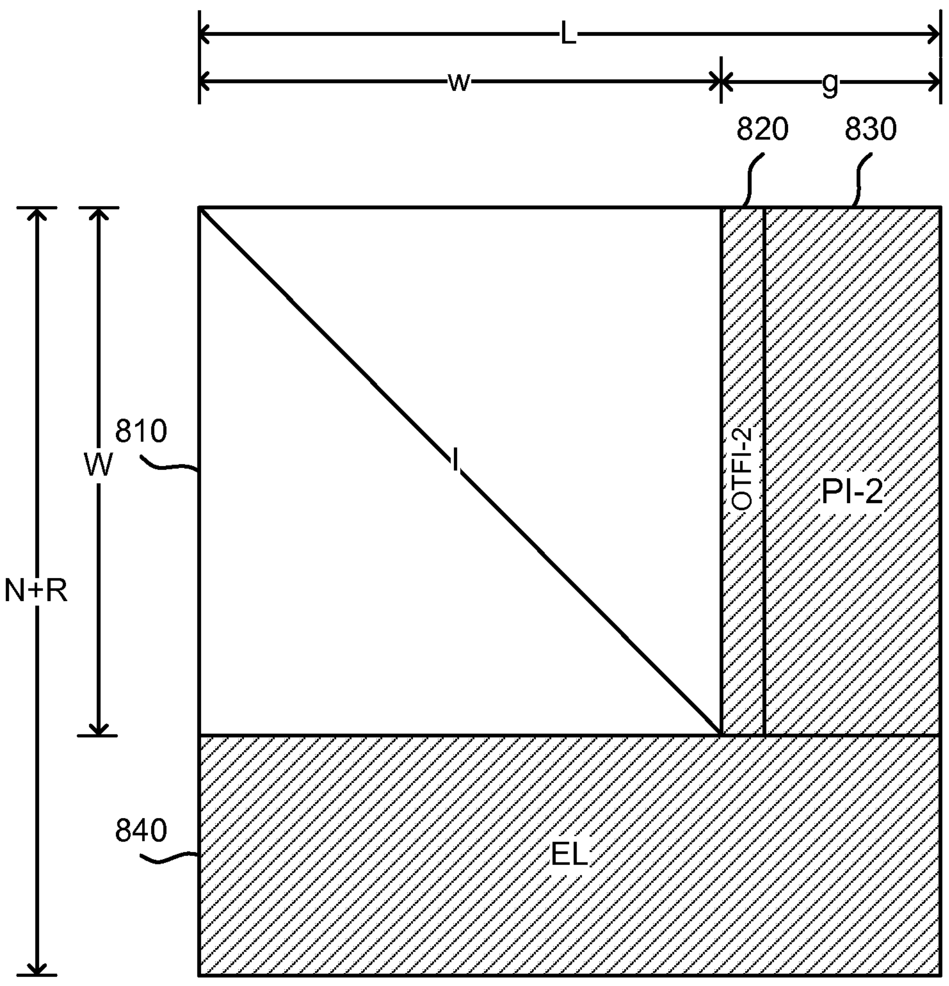


Fig. 24

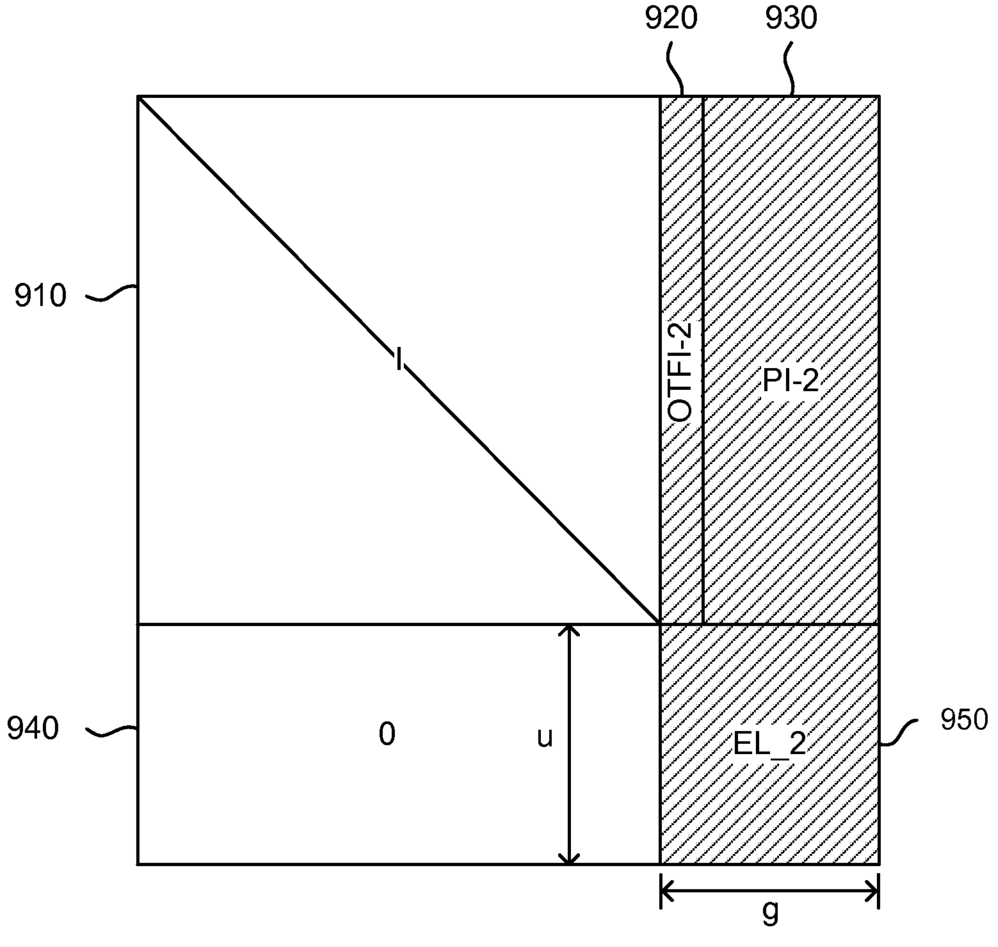


Fig. 25

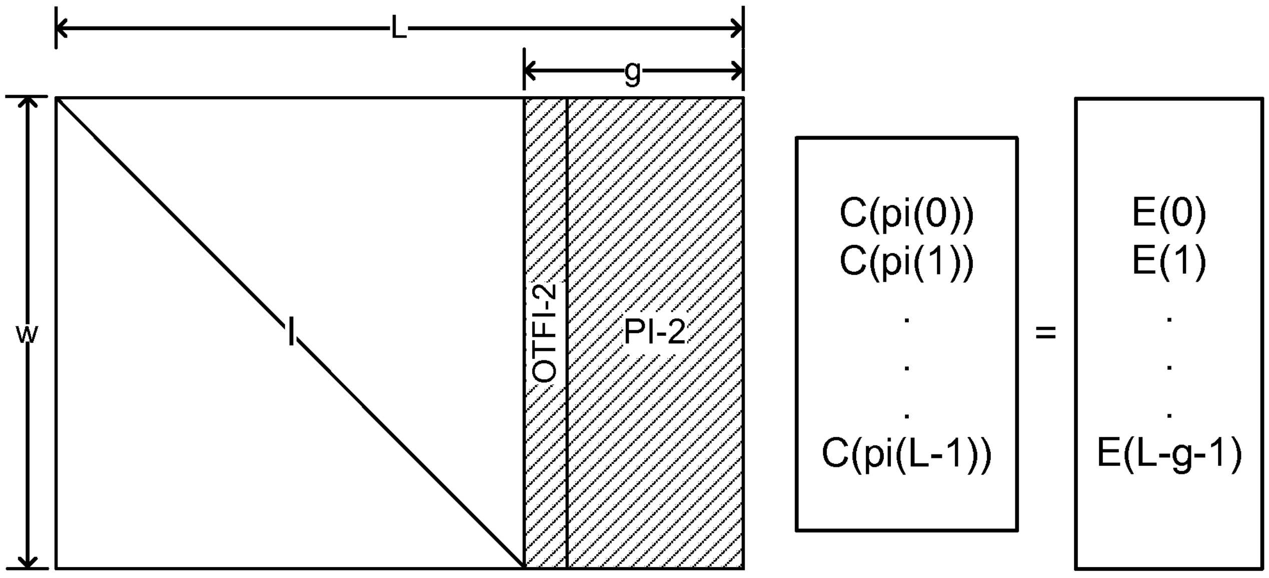


Fig. 26

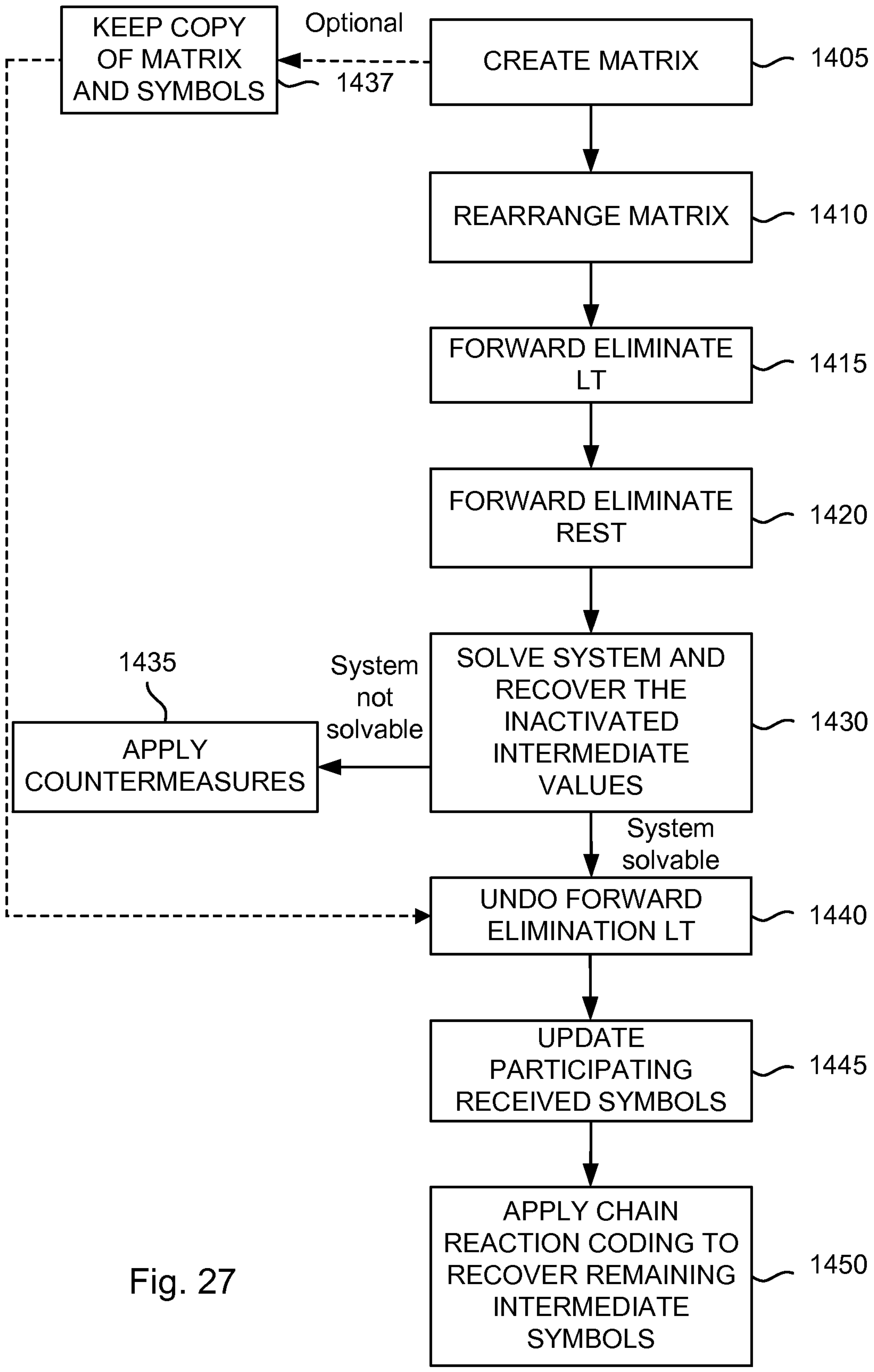

KEEP COPY OF MATRIX AND SYMBOLS
Optional
1437
CREATE MATRIX
1405
REARRANGE MATRIX
1410
FORWARD ELIMINATE LT
1415
FORWARD ELIMINATE REST
1420
1435
System not solvable
SOLVE SYSTEM AND RECOVER THE INACTIVATED INTERMEDIATE VALUES
1430
APPLY COUNTERMEASURES
System solvable
UNDO FORWARD ELIMINATION LT
1440
UPDATE PARTICIPATING RECEIVED SYMBOLS
1445
APPLY CHAIN REACTION CODING TO RECOVER REMAINING INTERMEDIATE SYMBOLS
1450

Fig. 27

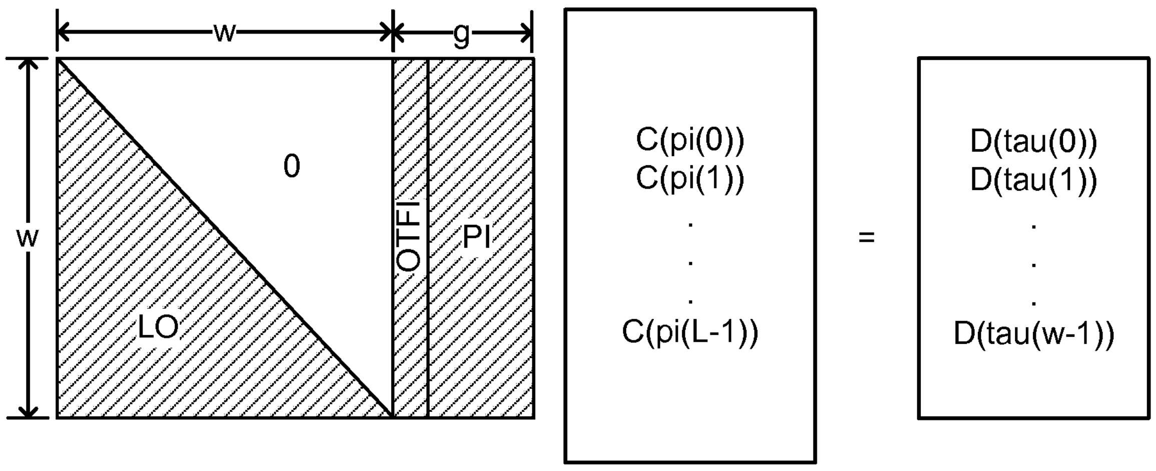

w
g
w
0
LO
OTFI
PI
C(pi(0))
C(pi(1))
.
.
.
C(pi(L-1))
=
D(tau(0))
D(tau(1))
.
.
.
D(tau(w-1))

Fig. 28

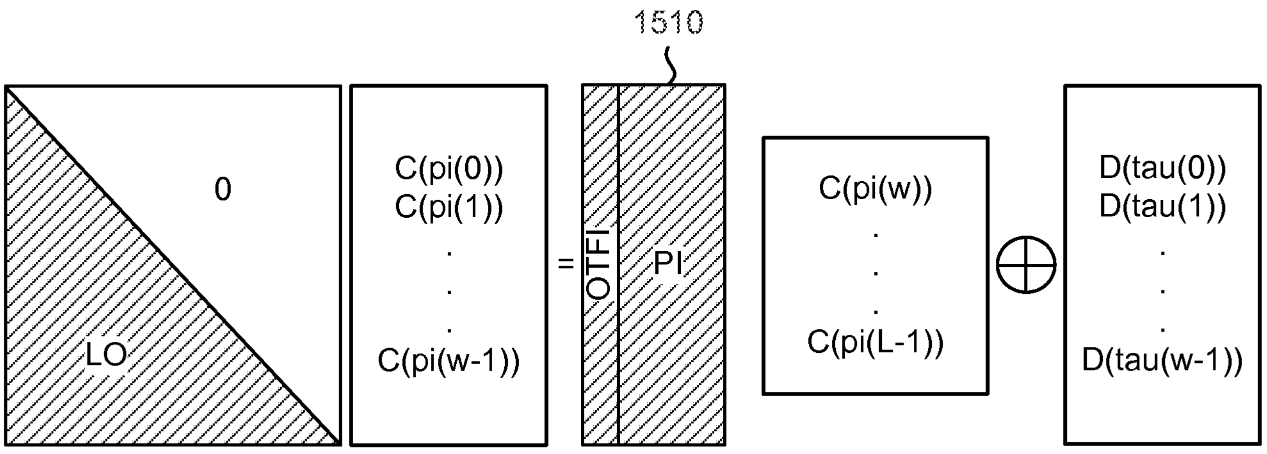

0
LO
C(pi(0))
C(pi(1))
.
.
.
C(pi(w-1))
=
1510
OTFI
PI
C(pi(w))
.
.
.
C(pi(L-1))
⊕
D(tau(0))
D(tau(1))
.
.
.
D(tau(w-1))

Fig. 29

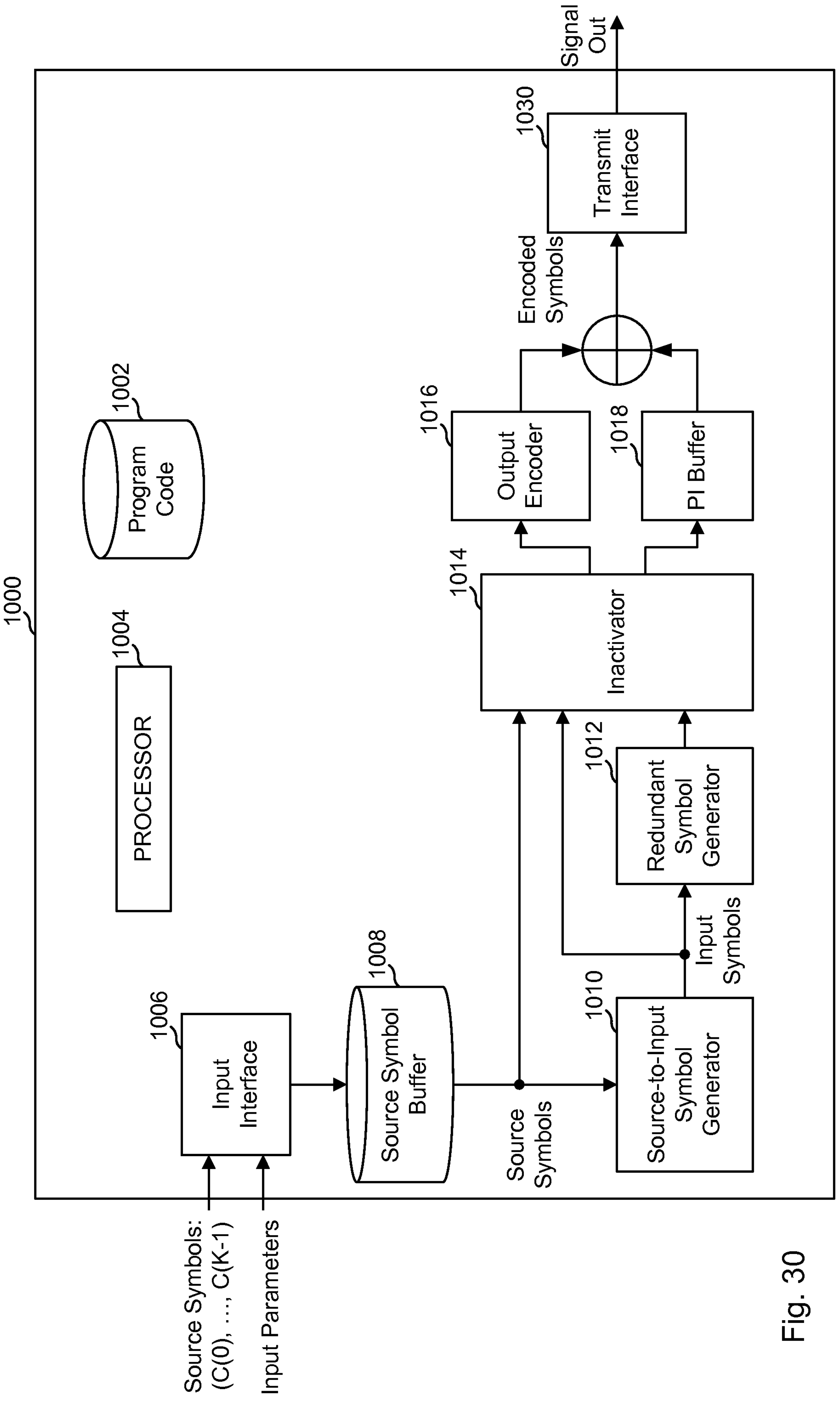

1000
1002
Program Code
1004
PROCESSOR
1006
Input Interface
Source Symbols: (C(0), ..., C(K-1)
Input Parameters
1008
Source Symbol Buffer
Source Symbols
1010
Source-to-Input Symbol Generator
Input Symbols
1012
Redundant Symbol Generator
1014
Inactivator
1016
Output Encoder
1018
PI Buffer
Encoded Symbols
1030
Transmit Interface
Signal Out

Fig. 30

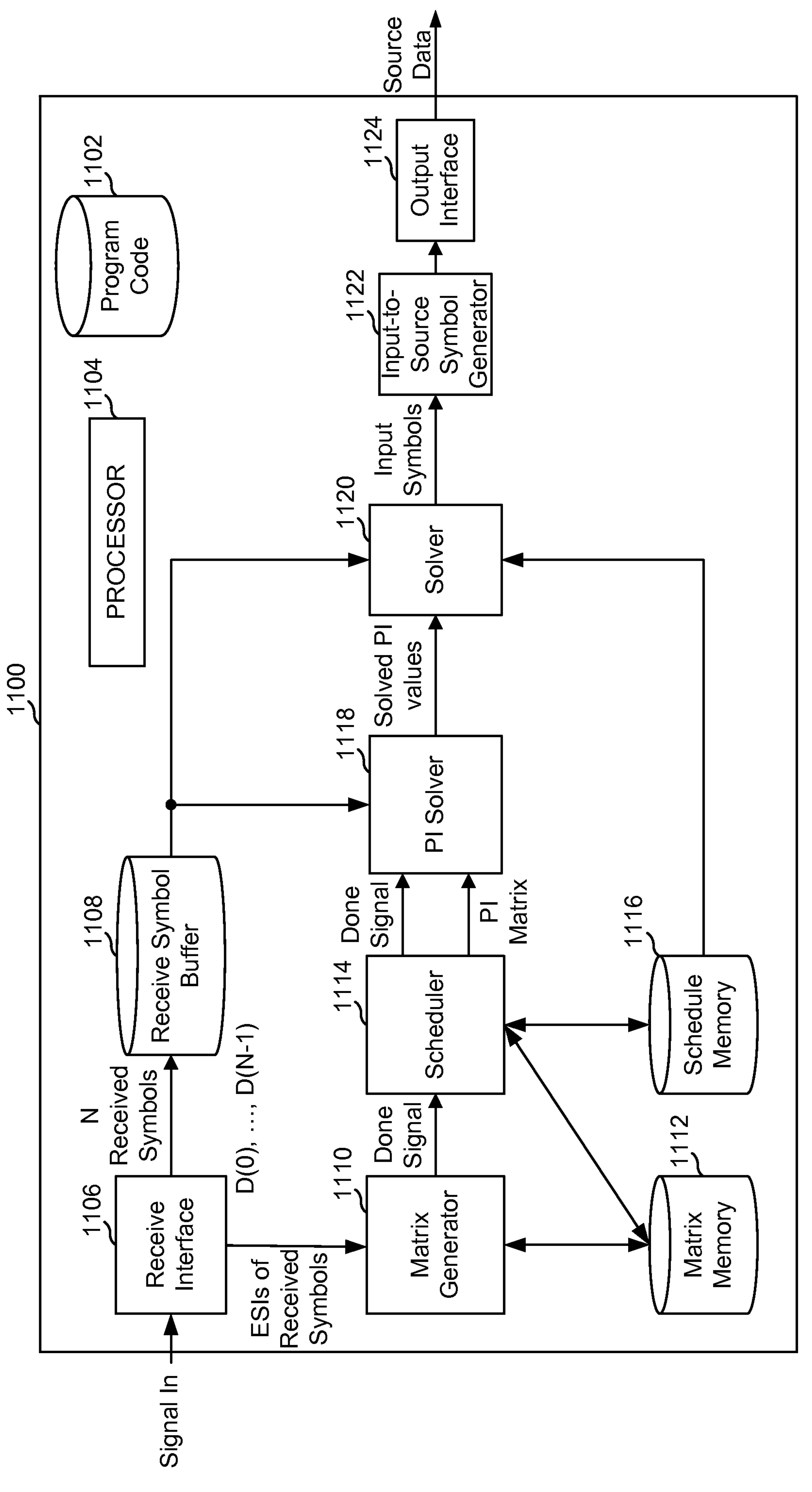

1100
1102
Program Code
1104
PROCESSOR
Signal In
1106
Receive Interface
N Received Symbols
1108
Receive Symbol Buffer
D(0), ..., D(N-1)
ESIs of Received Symbols
1110
Matrix Generator
Done Signal
1112
Matrix Memory
1114
Scheduler
Done Signal
PI Matrix
1116
Schedule Memory
1118
PI Solver
Solved PI values
1120
Solver
Input Symbols
1122
Input-to-Source Symbol Generator
1124
Output Interface
Source Data

Fig. 31